Energistics Transfer Protocol (ETP) Specification v1.1

Standards Document Title	Energistics Transfer Protocol (ETP) Specification
Version of Standard	**1.1**
Abstract	Energistics Transport Protocol (ETP) is a data-exchange specification that enables the efficient transfer of real-time data between applications. Initially conceived as part of the WITSML specification, ETP is now a component in the Energistics Common Technical Architecture and can be used with other Energistics specifications, such as PRODML and RESQML.
Prepared by:	WITSML SIG/ETP Work Group
Date published:	31 October 2016
Document type:	Specification
Keywords:	Standards, data, information, process

Document Information	
Document Version	1.0
Date	31 October 2016
Language	U.S. English

Acknowledgements

Energistics would like to thank members of the WITSML Special Interest Group and the Energistics Architecture team.

Amendment History			
Version	**Date**	**Comment**	**By**
1.0	30 June 2015	First commercial publication of the ETP specification. (Previous releases were “technology previews” intended for review, testing, and feedback.)	Energistics and ETP Work Group of the WITSML SIG
1.1	31 Oct 2016	Commercial release in support of version 2 of Energistics domain standards.	Energistics and ETP Work Group of the WITSML SIG

Table of Contents

1 Introduction

Energistics Transport Protocol (ETP) is a data exchange specification that enables the efficient transfer of real-time data between applications. 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, which includes WITSML, PRODML, and RESQML. Initially designed to be part of the WITSML specification, ETP is now part of the Energistics Common Technical Architecture (CTA).

One of the goals of ETP is to replace TCP/IP WITS level 0 data transfers with a more efficient and 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.

ETP is being expanded beyond real-time data transfer to include functionality for data discovery and historical data queries. It has been designed to work with multiple data models; for example, ETP is now the underlying protocol for WITSML v2.0 (replacing the SOAP API of previous versions of WITSML) and can be used to transfer data from WITSML v1.4.1.1, RESQML, and PRODML. (Separate implementation specifications will explain how to use ETP with these various data models.)

1.1 Document Details: Audience, Purpose, Scope

This document is intended for IT and software professionals who want to implement ETP. A basic understanding of the technology and related general concepts is assumed, though some contextual information is provided.

1.1.1 This Document is Created from the ETP UML Model

ETP has been designed and developed using UML® implemented with Enterprise Architect (EA), a UML modeling tool from Sparx Systems. The document contains these basic types of content:

- **Normative**: Content generated directly from the UML model. Normative content is the required information that defines mandatory behaviors and rules of ETP.
- **Non-normative**: Supplemental explanatory content (like this introduction chapter). Non-normative content provides necessary context for understanding ETP and how it works. It does not define specification behaviors.

1.1.2 Stability Index

Chapter 3 of this document indicates the stability of each ETP protocol, using values described in the table below. This concept of assigning a stability index 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.

Stability Index		
Index No.	**Status**	**Description**
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.
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 core team.
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.
3	**Stable**	The API has proven satisfactory, new features will not be added. Backwards compatibility is guaranteed, unless a critical issue is found.
4	**Locked**	The API will not change. Please do not suggest changes in this area; they will be refused.

1.1.3 Resource Set

ETP includes the following resources as part of the standard download, available on the Energistics website: http://www.energistics.org/standards-portfolio/energistics-transfer-protocol.

	Document / Resource	**Description**
1.	*ETP Specification* (This document.)	Defines content, messages, and behaviors of ETP.
2.	*Energistics Identifier Specification*	Describes the syntax and semantics of data object identifiers as used within the Energistics family of data-exchange standards.
3.	ETP UML Data Model	The UML data model that developers and architects can explore for better understanding of data objects, definitions, organization, and relationships, in context. Used to generate the Avro schemas and the normative parts of this specification. Energistics saves the UML model as an XMI file, a format that can be imported by any UML data modeling tool.
4.	Schemas	Avro schemas as described in this document.

1.1.4 Documentation Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in RFC 2119. (http://www.ietf.org/rfc/rfc2119.txt).

1.2 Overview of ETP and How it Works

This section is non-normative. It provides an overview of how ETP works, specifically it:

- Defines protocols and subprotocols and how ETP works with existing standard communication protocols. See Section 1.2.1 (page 8).
- Describes the message transport mechanism, WebSocket protocol. See Section 1.2.2 (page 9).
- Describes how messages are formatted and serialized using Avro. See Section 1.2.3 (page 10).
- Describes roles in the message exchange. See Section1.2.4 (page 10).

1.2.1 Protocols and Subprotocols

ETP is a **communication protocol**. Generically, a communication protocol can be thought of as a precise set of rules for the exchange of data between agents, usually in the form of messages. ETP messages:

- Are transported using the WebSocket protocol.
- Are defined in schemas and serialized using Avro (a system specifically designed for this purpose).

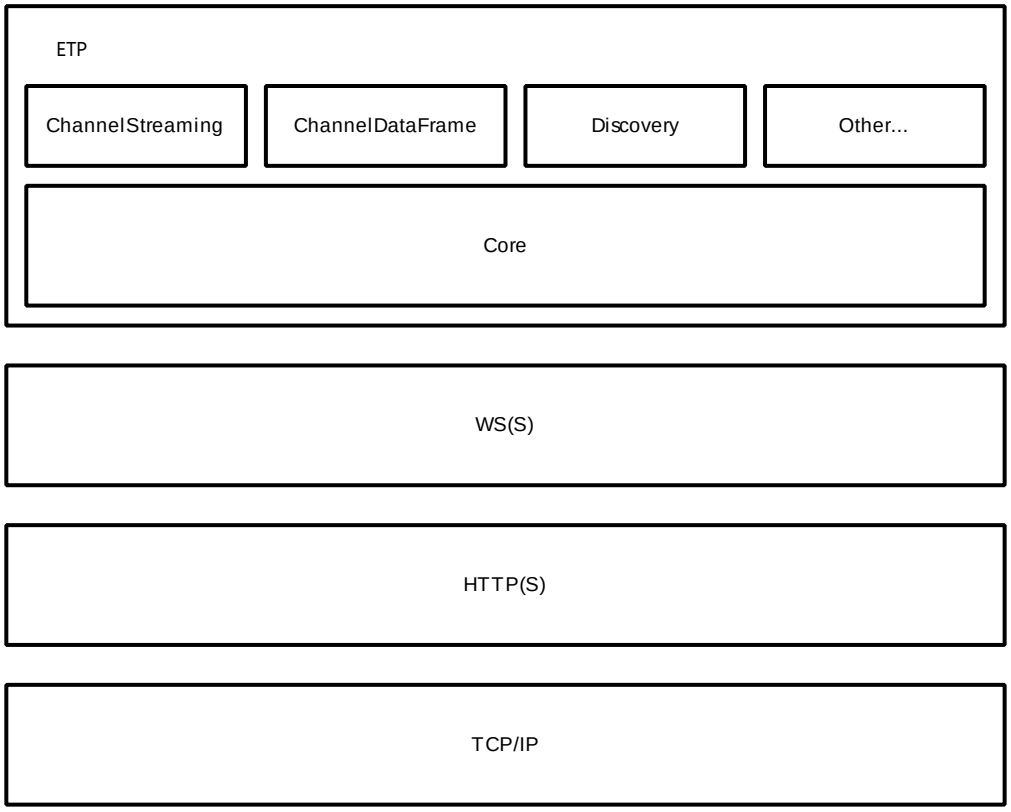

Figure 1: ETP Protocol Stack

Like most modern communication protocols, ETP uses a layered approach and sits on top of the existing Transmission Control Protocol (TCP) layered model (Figure 1: ETP Protocol Stack). Thus, the concept of protocol is used in many contexts throughout this document, and the notion of a **subprotocol** is used to discuss protocols that sit (somewhat un-intuitively) just ***above*** another protocol in the stack.

We can see that ETP is itself a subprotocol of the WebSocket protocol and that ETP also has its own layers and subprotocols, each designed to carry specific data that follows a specific pattern (in terms of size, frequency, and variability). The Core protocol has a direct connection to WebSocket and is agnostic to the various kinds of messages that are carried in each of its own subprotocols. This layered approach allows for separation of concerns between the various parts of the stack and supports the adoption of future standards that may be developed lower in the stack.

IMPORTANT! In this document, the terms *protocol* and *subprotocol* are used interchangeably, about any layer, depending on context.

1.2.1.1 Identifying ETP Protocols

Each of the ETP protocols is assigned a numeric identifier as part of this specification, and all messages are identified with this number 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 is used regularly, to allow for the smallest, most efficient binary transfers.

1.2.1.2 ETP Message Approach

ETP has been designed to support multiple patterns, or styles of message transfer, especially with respect to message size, frequency, and complexity. Separate subprotocols of ETP are generally related to specific kinds of messages. This practice 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, the goal is 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 focused on high-speed streaming of channel data as described below.

1.2.2 WebSocket Transport

ETP is designed to use the WebSocket protocol for transport. A full description of WebSocket is beyond the scope of this document. In brief, WebSocket is a protocol, standardized by the Internet Engineering Task Force (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) using TCP and the standard HTTP(s) ports 80/443. This approach allows communications to easily and safely cross many corporate firewalls. 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:

- ETP is considered a subprotocol of WebSocket as defined in sections 1.9 and 11.5 of RFC 6455.
- ETP messages map directly to the “payload data” section of WebSocket frames and messages. In most cases, these details are invisible to developers, because developers use a vendor-supplied library to interface with WebSocket.

All ETP communication is carried out through the **asynchronous** exchange of **messages**. This approach is distinct from the request/response pattern normally associated with HTTP, and the “RPC style” associated with many SOAP implementations, including WITSML 1.*. Of course, many use cases still require a request on the part of one agent and expect a response of some sort from the other; however, implementers should always consider these things to be happening asynchronously and handle processing using state machines that model the various timings of message exchange that could occur.

The WebSocket protocol guarantees the delivery of messages in the same order that they were dispatched. To ensure messages are correlated correctly, ETP uses several mechanisms, which include:

- All messages within a session are numbered. Message numbers are integers and MUST be unique within a session.
- A correlationId is included in each message, which may designate a given message as being part of a ‘response’ to a previous request.
- Large messages may be sent in parts, with a mechanism in the message header for determining when the last part arrived.

- Various ETP subprotocols may impose specific ordering and numbering of certain messages.

These mechanisms are described in more detail below.

1.2.3 Avro Serialization

The serialization of messages in ETP follows a subset of 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. The ETP Workgroup selected Avro after a review of several similar serialization systems. Again, ETP uses only a subset of the Avro functionality as described here:

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

The Avro specification supports the use of both **binary** and **JSON** (JavaScript Object Notation) encoding of data. ETP also supports the use of both, with the following caveats:

- **All messages within a given ETP session must use the same encoding (binary or JSON)**. The encoding that is used is negotiated as described in the discussion below of Protocol 0.
- **Agents are not required to support both encodings.** This exception 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 release of ETP, it is assumed that all parties have prior knowledge of the schemas involved. In future releases, capabilities will be added to exchange version-specific schemas at the time of negotiating the session, which will allow an agent to consume any ETP message, even if it cannot use all of the information in the message.

1.2.3.1 Serialization of URIs

Many of the messages in ETP make use of the URIs as opaque references to objects in a system. The definitions of these URIs conform to the specification in RFC 3986 (http://tools.ietf.org/html/rfc3986). Further definition of the specifics of Energistics standard URI schemes are found in the *Energistics Identifier Specification*. Of particular note for the serialization of these URIs in Avro/ETP:

- Sending agents MUST urlencode (i.e., percent encode, as per section 2 of RFC 3986) any components that contain reserved characters for the URI scheme, the specific reserved characters being found in the *Energistics Identifier Specification*.
- Receiving agents MUST decode all incoming URIs.

1.2.4 Client, Servers, and Roles

Consistent with all TCP communication, ETP includes the fundamental roles of **client** and **server**. That is, in all ETP communication, one agent must be a server that is listening on a TCP port, and one agent must be a client that begins by connecting on that address and port. However, beyond these basic roles, the general direction of information flow is independent of this client/server relationship. This scenario is a business requirement for WITSML.

To accommodate this requirement, each subprotocol in ETP defines a set of roles that are appropriate for that protocol. The assignment of these roles to agents happens as a part of the Core protocol (Protocol

0). In general, the client begins by telling the server which subprotocols it wants to use and the named roles it wants the server to fulfill in this session. The server then responds to indicate it is able to fulfill its role; if it cannot, it sends an exception message to client. This “client-goes-first” logic is predicated on the basis that a client knows why it is connecting to a server; whereas, a server that is capable of supporting two different roles has no way of knowing which one the client wants to use.

For a more concrete example: The ChannelStreaming protocol (Protocol 1) defines two roles: **producer** and **consumer**. Each agent is either a producer or a consumer of streaming data. Simplistically, we could say that a measurement device or rig aggregator is usually a producer; a service company WITSML store can be a either a producer or a consumer, and a Web browser or desktop client 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, and the aggregator as client would ask to be a producer.

1.3 Overview of ETP Protocols

The current version of ETP includes the protocols listed in the table below. For the actual normative content that defines the protocols, see Chapter 2 (page 14) and Chapter 3 (page 39).

Message numbers 1000-1999 are used in multiple protocols, such as for errors and acknowledgements.

Protocol Number and Name	Description	Stability Index
Protocol 0: Core (For more information, see Section 1.3.1.1 (page 12).)	Creates and manages ETP sessions.	**3 - Stable**
Protocol 1: ChannelStreaming (For more information, see Section 1.3.1.2 (page 12).)	Defines a set of messages for exchanging channel-oriented data, where a channel is a time or depth series of individual data points	**3 - Stable**
Protocol 2: ChannelDataFrame	Transfers arrays of channel data as data frames, where a frame is an array of values for a given set of channels aligned at a common index point.	**1 - Experimental**
Protocol 3: Discovery	Uses a RESTful (representational state transfer) approach to enable store customers to enumerate and understand the contents of store of data objects.	**3 - Stable**
Protocol 4: Store	Used to perform CRUD operations (create, retrieve, update and delete) on data objects in a store. The roles for the Store protocol are “store” and “customer”.	**2 - Unstable**
Protocol 5: StoreNotification	Used to allow store customers to receive notification of changes to data objects in the store in an event-driven manner.	**2 - Unstable**
Protocol 6: GrowingObject	Used to manage the growing parts of data objects that are index-based (i.e., time and depth) but are not appropriate for the Channel Streaming protocol.	**2 - Unstable**
Protocol 7: DataArray	Used to transfer large, binary arrays of heterogeneous data values.	**1 - Experimental**
Protocol 8: WitsmlSoap	Allows ETP clients to make WITSML 1.2, 1.3, and 1.4 server calls using ETP instead of SOAP. Used to support earlier versions of the WITSML specification over ETP.	**1 - Experimental**
Protocol 9-1999	Undefined. Reserved for future use.	NA
Protocol 2000+	Custom. Available for custom use (not Energistics).	NA

1.3.1.1 Protocol 0: Session Management (or Core Protocol)

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**.

When a session is created, it is assigned a universally unique identifier (UUID). 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 approach allows for much smaller messages on the wire and more efficient processing of messages in code. **Figure 2** shows the layout of an ETP message.

Figure 2: ETP message layout.

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

- To identify the message subprotocol within ETP.
- To identify the type of message being sent (messageType), which implicitly defines the schema for the message data.
- To identify the ID of the message itself. Each message in an ETP session is uniquely numbered. The number is unique within the context of the session and agent type (client/server).
- The message header contains an optional correlationId attribute. Its purpose is somewhat context-sensitive, depending on the specific message, but in general it is used to correlate messages that are in response to some previous request. For example, when a consumer makes ChannelRangeRequest, all of the resulting ChannelData messages will have a correlationId that is the value of the original ChannelRangeRequest message.
- Finally, each message has an optional messageFlags attribute. This attribute acts as a bit-field and allows multiple Boolean flags to be set on the message. Three flags are currently defined:
  - One that says this is a multi-part response (i.e., more data messages are coming related to this request).
  - One that identifies a message as the final part of a multi-part message.
  - One that identifies a message as ‘NoData’.

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

1.3.1.2 Protocol 1: Channel Streaming

Protocol 1, named the **ChannelStreaming protocol**, is used to stream channel-oriented data from producers to consumers. A **channel** is a named object in the system, identified by a Uniform Resource Identifier (URI), that produces a stream of data points, usually representing a series of measurements (although other kinds of data points, such as comments, can also be streamed). In WITSML terms, a channel is comparable to a log curve; more generally, it is comparable to a tag in a process historian. Channels organize their data points according to one or more channel indexes, usually in time or depth (or both).

The streaming of data begins with the consumer asking the producer to describe one or more channels using a supplied URI. This URI can represent a single channel, or a higher-level object, such as a log, that expands to several channel URIs. These potentially nested channels result in one or more **ChannelMetadata** messages being sent, which describe the nature of the channel (name, data type, mnemonic, units of measure, etc.). This message also assigns the channel a unique, integer-valued

ChannelId—which is valid only for the length of the session—and that is used in all subsequent references to the channel.

2 Protocol

This section is normative. It represents the published ETP specification and is generated directly from the UML model. This section provides a dynamic view of each ETP protocol using standard UML conventions. For each protocol, the following information is given:

- An **overview** of the protocol
- Its **requirements** (or general behavior)
- **Sequence diagrams** to model the normal flow of messages. Assume that each message also carries a message header.
- **Interfaces for agents** in the interaction (e.g., clients and servers, consumers and producers, etc.). Generally, each protocol has 2 interfaces, one for each role. In the future, this number of interfaces is not guaranteed, because some protocols may be purely peer-to-peer, with identical roles for 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.
- Interface methods that send messages are not stereotyped; methods that receive or handle incoming messages are stereotyped as <> methods. In some cases (such as CloseSession), either party may send or receive the message, and this is reflected by having both call and event methods.
- In some cases, there are **protocol state diagrams** to model the possible states of each role in a subprotocol.

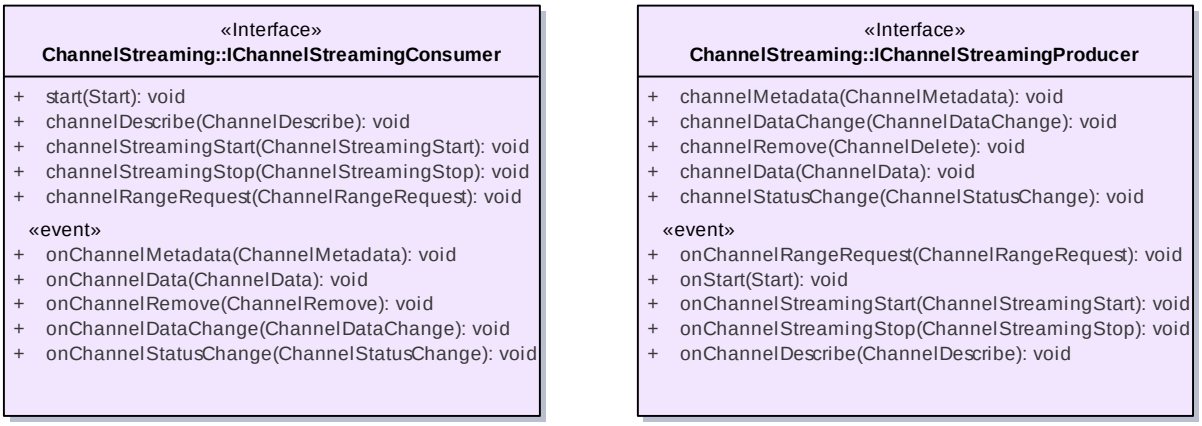

Figure 3: Sample Interfaces

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

2.1 Core

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

- Establishes the WebSocket connection.
- Manages session life cycle.
- Negotiates the subprotocols 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. When these messages are used, the protocol field in the message header will not necessarily be 0; it will be the appropriate value for the protocol that is using the common message (e.g., a ProtocolException on the Discovery protocol would use a value of 3).

2.1.1 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 and agent type. 3. Increasing within a multi-part response such that message ID order can be used to determine the order of the response. The IDs used by clients and servers are completely independent of one another. Put another way, the 'primary key' of any given message could be thought of as the sessionId + agentType + messageId. Note also that there is specifically no requirement for message IDs to be sequential or for any correlation between message IDs and any particular sub protocol.
Protocol Negotiation	Behavior	1. In the RequestSession message, the client specifies the protocols, versions, and roles which it expects the **server** to support for this session. 2. If the server supports at least one of the requested protocols, then the server MUST respond to this message using the OpenSession message, indicating which of the requested protocols and roles it can support. 3. If the server supports none of the requested protocols, it MUST send a ProtocolException with the ENOSUPPORTEDPROTOCOLS error code. 4. The server MAY offer to support only some of the requested protocols. It MUST not offer to support any additional protocols. 5. The server MUST NOT change the requested roles in its response. 6. The server MAY change the supported version of a protocol. 7. If the server is able to support the version the client requests, it MUST do so. 8. If the server response does not provide adequate functionality, then the Client MAY CloseSession immediately.
Session State	Behavior	1. The definition of a session is all messages associated with a given session UUID, including the original RequestSession. 2. Clients and servers are responsible for maintaining session state, as defined by each supported subprotocol. 3. For protocol 0, the session state includes: a) The session UUID. b) The current highest-valued message IDs (for client and server). c) The state of currently-active multi-part messages. d) All of the SupportedProtocol information from OpenSession.
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 received. Specific information included in

Requirement	Type	Description
		session state MAY be define at the subprotocol level. 2. Server MUST maintain this state for **1hr**, +/- 1 minute after disconnect. 3. On reconnection, the client MAY include the **etp-sessionid** HTTP header in the upgrade request. This effectively is a request to re-activate that session. In this instance, the RequestSession messageId MUST be valid (i.e., a continuation of the ID sequence) from that session. 4. If the session is valid, the server MUST include the identical etp-sessionid HTTP header in the upgrade response. 5. If the session is not valid, the 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.
Token Expiry	Behavior	As discussed in Section 4.2 of this specification.
URIs	Behavior	All URIs MUST be URL-encoded.

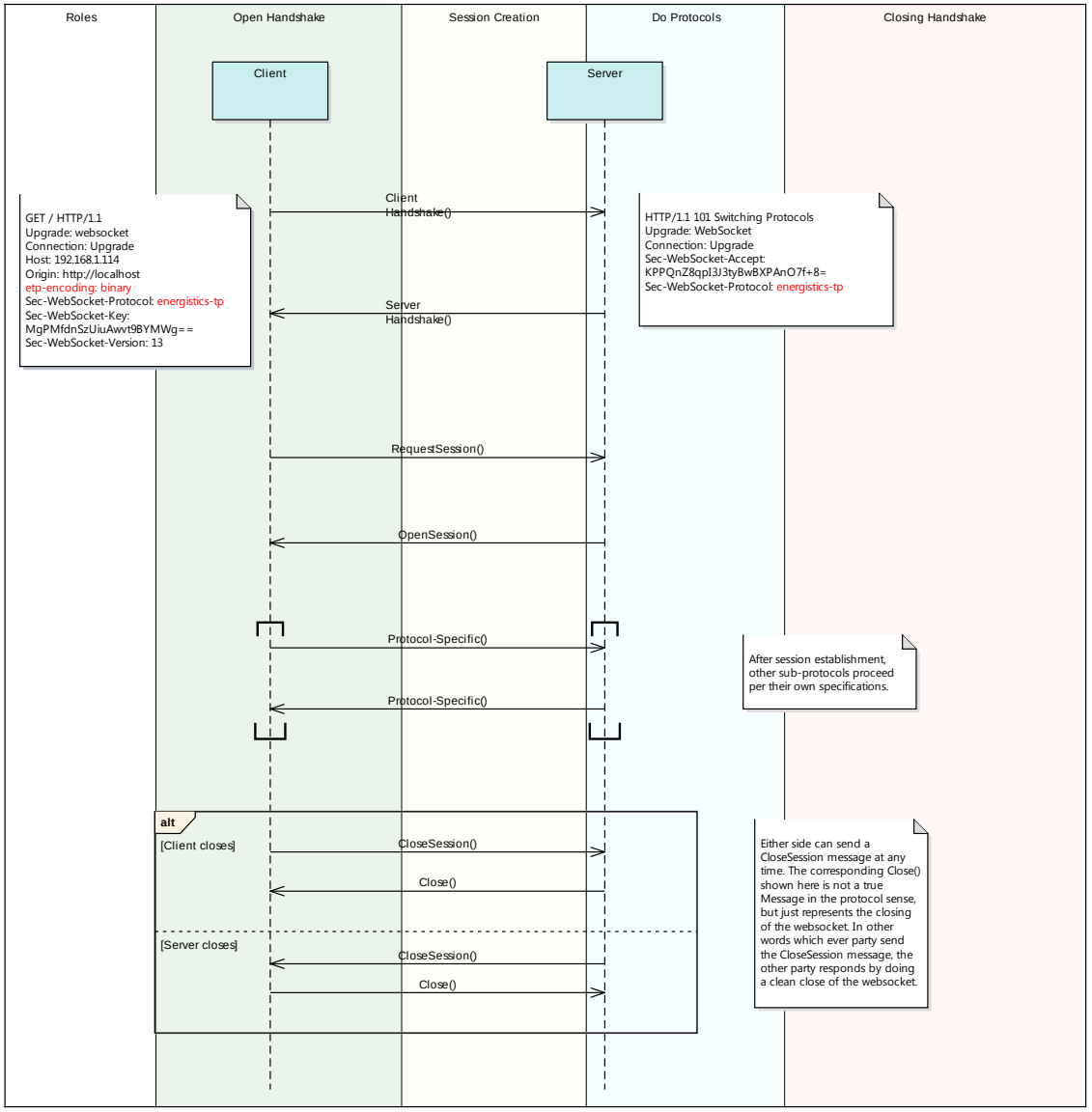

Figure 4: Protocol 0

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

1. A client begins with the standard WebSocket handshake and the server responds by doing the upgrade from HTTP to WebSocket. For more information on the handshake, see RFC 6455 (https://tools.ietf.org/html/rfc6455) .

Specific to ETP, the client MUST specify the *Sec-Websocket-Protocol* header value of **energistics-tp**, and the server MUST reply with the same. The client MAY also supply the custom header of *etp-encoding* with a value of **binary** or **json**, which 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 cannot currently add custom headers to a WebSocket 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 MUST 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.

1. A client then sends a RequestSession message. This message consists of a list of subprotocols that the client intends to use on this connection, along with a key/value map of configuration parameters for each subprotocol. The names of allowable keys are strictly controlled by this specification and defined for each subprotocol.
2. The requested protocols are started and message passing begins. The sequence and interaction diagrams for each are in the subprotocol sections below.
3. If the server cannot support all of the requested protocols, it MUST respond with a list of the requested protocols that it **does** support, leaving it up to the client to decide whether to continue with the session.

2.1.2 Interface: IClient

IClient represents the interface that MUST be implemented from the client side of Protocol 0.

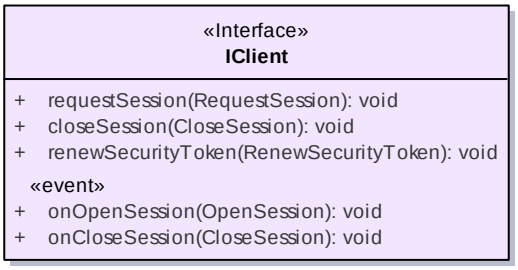

Figure 5: IClient

2.1.2.1 Methods

Method	Description	Type	Parameter Summary
requestSession	Requests a new ETP session with a server.		(in) eventData: RequestSession
onOpenSession	Handles OpenSession event from a server.	event	(in) eventData: OpenSession

Method	Description	Type	Parameter Summary
closeSession	Sends CloseSession to a server.		(in) eventData: CloseSession
onCloseSession	Handles CloseSession event from a server. Note that IClient can both send and receive CloseSession.	event	(in) eventData: CloseSession
renewSecurityToken	Sent from a client to a server to refresh the security token.		(in) eventData: RenewSecurityToken

2.1.2.2 State Machine

The following diagram shows the definition of Protocol 0 as a state machine, from the standpoint of a client. This diagram incorporates both the connection/disconnection with WebSocket, 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 standpoint, there may be more complex state machines created that combine the subprotocols. For example, an application may use the Discovery protocol to interactively determine what channels are available for streaming, and then use the ChannelStreaming protocol to stream data on those channels. These higher-level constructs are not specified as part of the protocol.

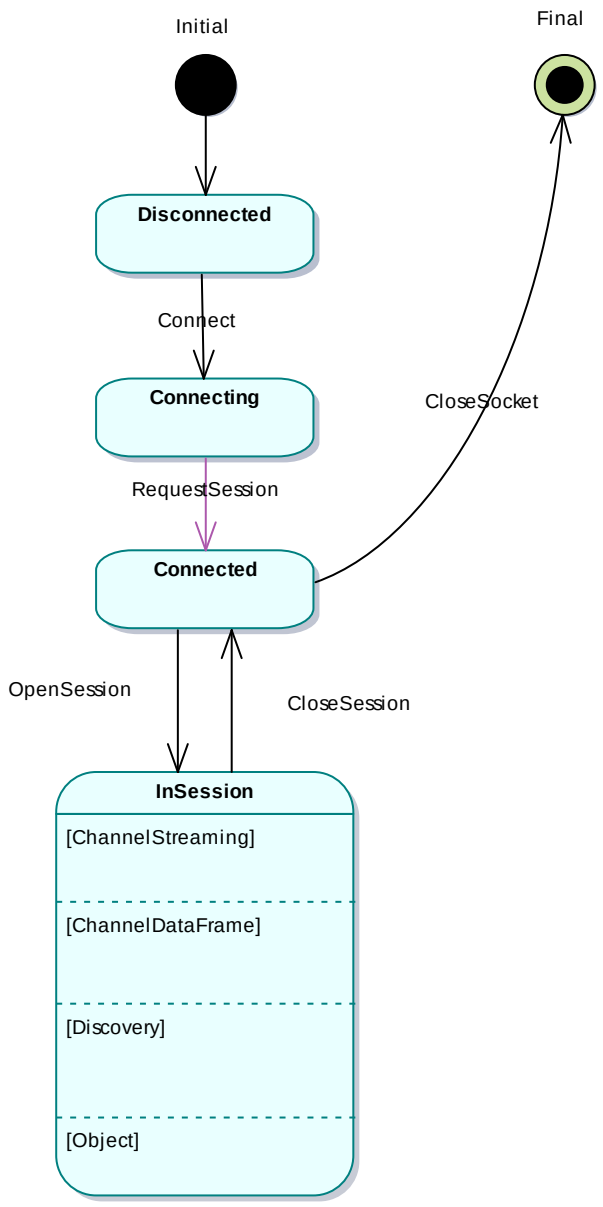

Figure 6: ClientStateMachine

2.1.3 Interface: IServer

IServer represents the server end of the interface that MUST be implemented for Protocol 0.

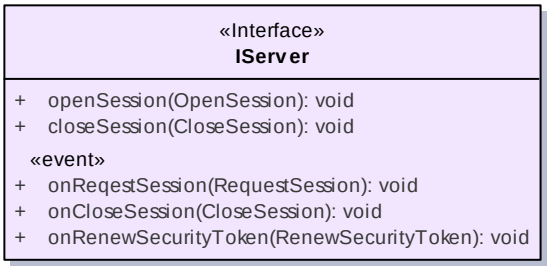

Figure 7: IServer

2.1.3.1 Methods

Method	Description	Type	Parameter Summary
onReqestSession	Handles RequestSession events from a client.	event	(in) eventData: RequestSession
openSession	Opens a new ETP session with a client.		(in) eventData: OpenSession
closeSession	Sends CloseSession to a client.		(in) eventData: CloseSession
onCloseSession	Handles CloseSession events from a client. Note that IServer can both send and receive CloseSession.	event	(in) eventData: CloseSession
onRenewSecurityToken	Handles a renewed security token from the client.	event	(in) eventData: RenewSecurityToken

2.1.3.2 State Machine

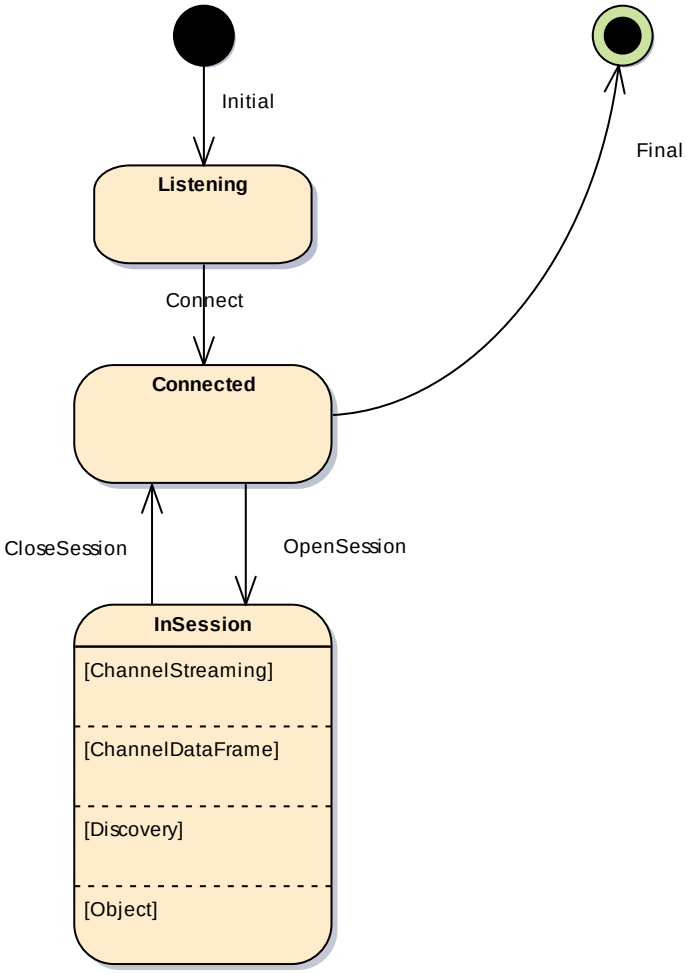

Figure 8: State Machine

2.2 ChannelStreaming

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.

2.2.1 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. Index order is always as appropriate for the IndexDirection of a Channel. 4. The same primary index value MUST NOT appear more than once in any ChannelData record (unless it is correlated to ChannelRangeRequest).
Range Requests	Behavior	1. The from and to indexes in a ChannelRangeRequest MUST be in index order. 2. The from and to indexes in a ChannelRangeRequest MUST always reference the primary index.
Session Survivability	Behavior	For the ChannelStreaming protocol: 1. Channel Metadata (including channelId) remains valid, provided that all ChannelMetadata records have been received (remembering ChannelMetadata can be a multipart response). 2. Multipart responses to ChannelRangeRequest messsages are not completed on session re-connect. If the consumer has not received all parts to a response, it MUST re-issue the ChannelRangeRequest. On re-connect, a producer MUST send all ChannelDataChange, ChannelStatusChange, and ChannelRemove messages that would have been sent had the connection remained alive. The consumer MAY receive a duplicate message.

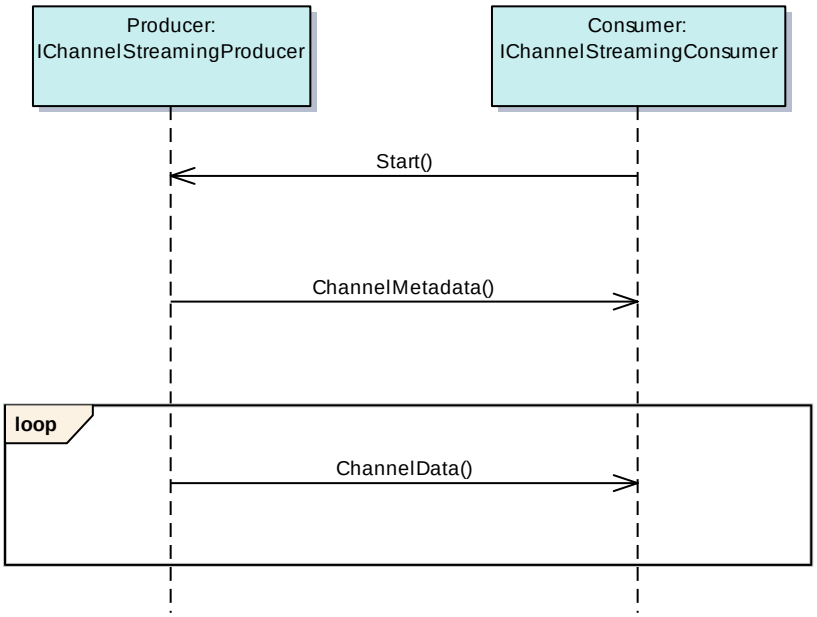

Figure 9: Protocol 1 - Simple Streaming

The simple streaming diagram shows the minimal session interaction for a simple producer of data (Protocol 1). This scenario 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, which indicates it does not accept requests to stream individual channels but always sends all of its channels. The name of the variable is *SimpleStreamer* and it MUST have a Boolean value of **true**.
- The producer MUST NOT send any data until the Start 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 sends at least one ChannelMetadata 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.
- After this, the producer can begin streaming ChannelData.

When a producer identifies itself as a SimpleStreamer, the producer and the consumer MUST NOT use any messages other than Start, ChannelMetadata and ChannelData. To avoid adding complexity to a protocol option that is designed to simplify producers, any use of unsupported messages in a SimpleStreamer session are ignored (i.e., if those messages are used, there is no requirement that the producer or consumer issue exceptions).

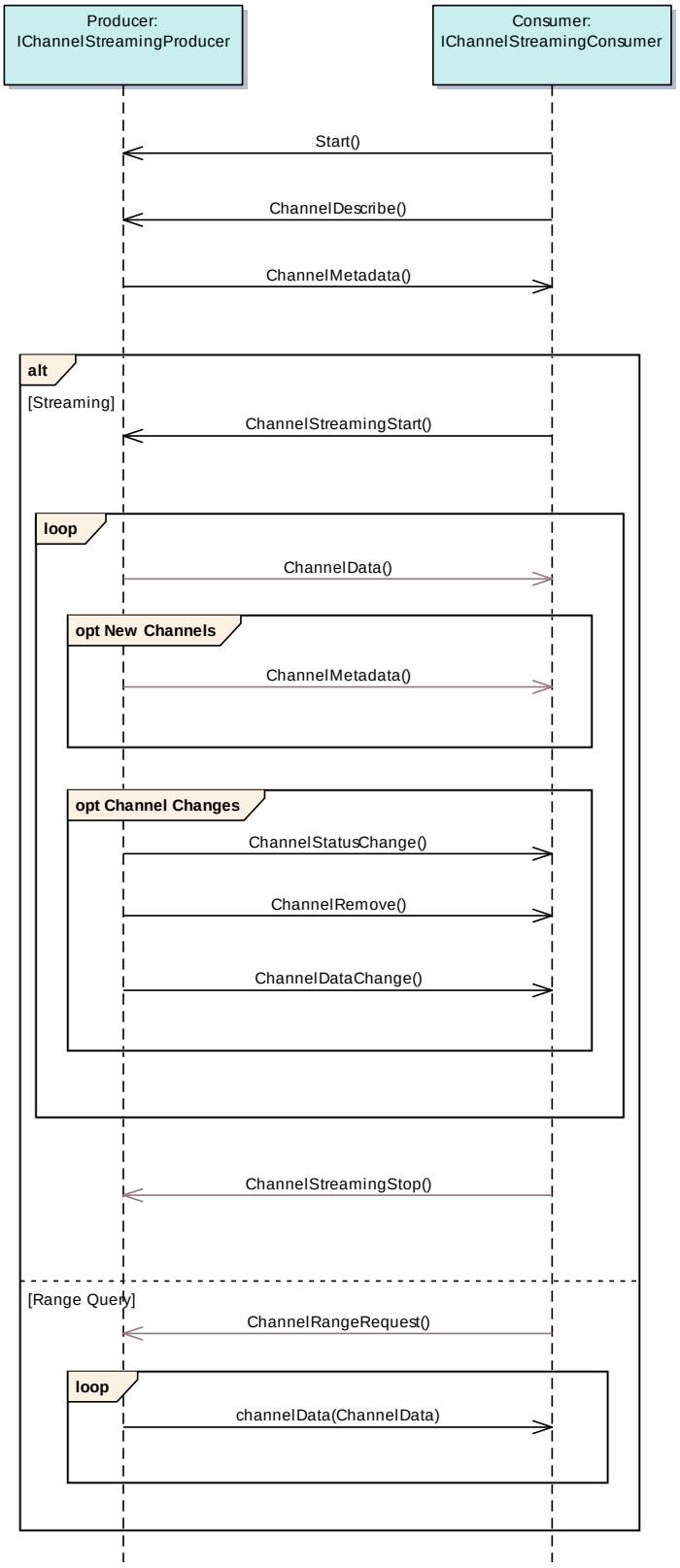

Figure 10: Protocol 1 - Channel Streaming

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

2.2.2 Interface: IChannelStreamingConsumer

IChannelStreamingConsumer defines the interface that MUST be implemented by the consumer role of the ChannelStreaming protocol.

2.2.2.1 Methods

Method	Description	Type	Parameter Summary
start	Start the protocol. For a SimpleStreamer, this message indicates it should start streaming data.		(in) eventData: Start
channelDescribe	Sent from a consumer to a producer to request metadata about one or more channels, specified by URI.		(in) eventData: ChannelDescribe
channelStreamingStart	Sent from a consumer to a producer to request that the producer begin streaming one or more channels.		(in) eventData: ChannelStreamingStart
channelStreamingStop	Sent from a consumer to a producer to request that streaming be discontinued on one or more channels.		(in) eventData: ChannelStreamingStop
channelRangeRequest	Sent from a consumer to a producer to request data over a specific range for one or more channels.		(in) eventData: ChannelRangeRequest
onChannelMetadata	Sent from a producer to aconsumer to describe channels that may be streamed to the consumer in future messages.	event	(in) eventData: ChannelMetadata
onChannelData	Contains an array of <index,value> 2-tuples for one or more channels.	event	(in) eventData: ChannelData
onChannelRemove	Sent from a producer to a consumer to indicate that a channel is no longer actively streaming data. When a channel has been removed, a producer MUST NOT send additional ChannelData messages on this channel. ChannelRemove is sent regardless of the value of receiveChangeNotification field in ChannelStreamingInfo record used to start this channel.	event	(in) eventData: ChannelRemove
onChannelDataChange	Sent from a producer to a consumer to notify the consumer of changed data points on the channel. Only sent if the receiveChangeNotification field in ChannelStreamingInfo record used to start this channel is set to True.	event	(in) eventData: ChannelDataChange
onChannelStatusChange	Sent from a producer to a consumer when the status of a channel (as defined in the status field of the ChannelMetadataRecord) has changed. Only sent if the receiveChangeNotification field in ChannelStreamingInfo record used to start this channel is set to True.	event	(in) eventData: ChannelStatusChange

2.2.2.2 Session

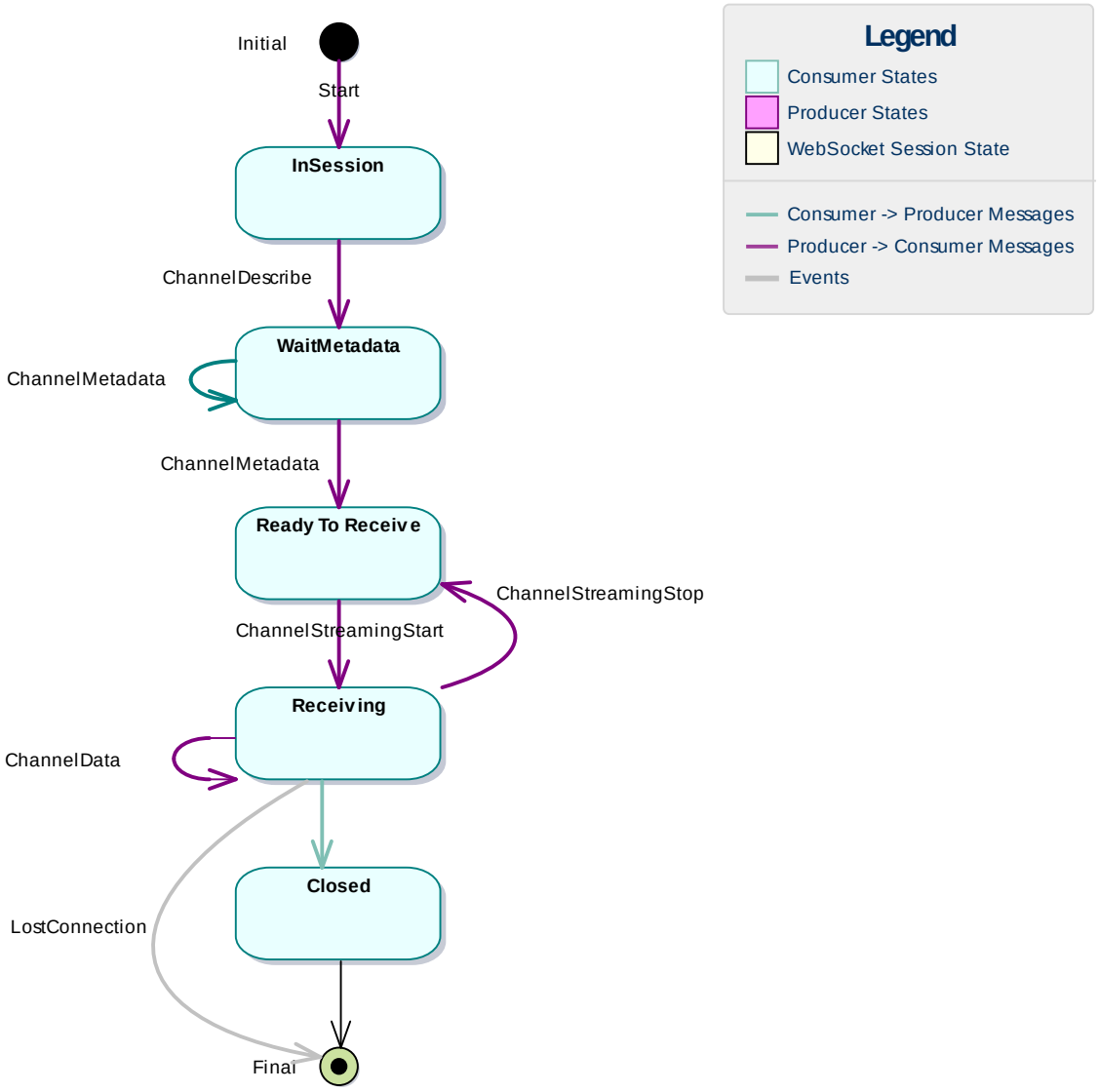

Figure 11: ETP Consumer as Client State

2.2.3 Interface: IChannelStreamingProducer

IChannelStreamingProducer defines the interface that MUST be implemented by the producer role of the ChannelStreaming protocol.

2.2.3.1 Methods

Method	Description	Type	Parameter Summary
channelMetadata	Sent from a producer to a consumer to describe channels that may be streamed to the consumer in future messages.		(in) eventData: ChannelMetadata
onChannelRangeRequest	Sent from a consumer to a producer to request data over a	event	(in) eventData: ChannelRangeRequest

Method	Description	Type	Parameter Summary
	specific range for one or more channels.		
onStart	Start the protocol. For a SimpleStreamer, this message indicates it should start streaming data. Includes parameters to throttle size and rate of messages.	event	(in) eventData: Start
onChannelStreamingStart	Sent from a consumer to a producer to request that the producer begin streaming one or more channels.	event	(in) eventData: ChannelStreamingStart
onChannelStreamingStop	Sent from a consumer to a producer to request that streaming be discontinued on one or more channels.	event	(in) eventData: ChannelStreamingStop
onChannelDescribe	Sent from consumer to a producer to request metadata about one or more channels, specified by URI.	event	(in) eventData: ChannelDescribe
channelDataChange	Sent from a producer to a consumer to notify the consumer of changed date points on the channel.		(in) eventData: ChannelDataChange
channelRemove	Sent from a producer to a consumer to indicate that a channel is no longer actively streaming data.		(in) eventData: ChannelDelete
channelStatusChange	Sent from a producer to a consumer when the status of a channel (as defined in the status field of the ChannelMetadataRecord) has changed. Only sent if the receiveChangeNotification field in ChannelStreamingInfo record used to start this channel is set to True.		(in) eventData: ChannelStatusChange
channelData	Contains an array of <index,value> 2 tuples for one or more channels.		(in) eventData: ChannelData

2.2.3.2 Channel Producer State

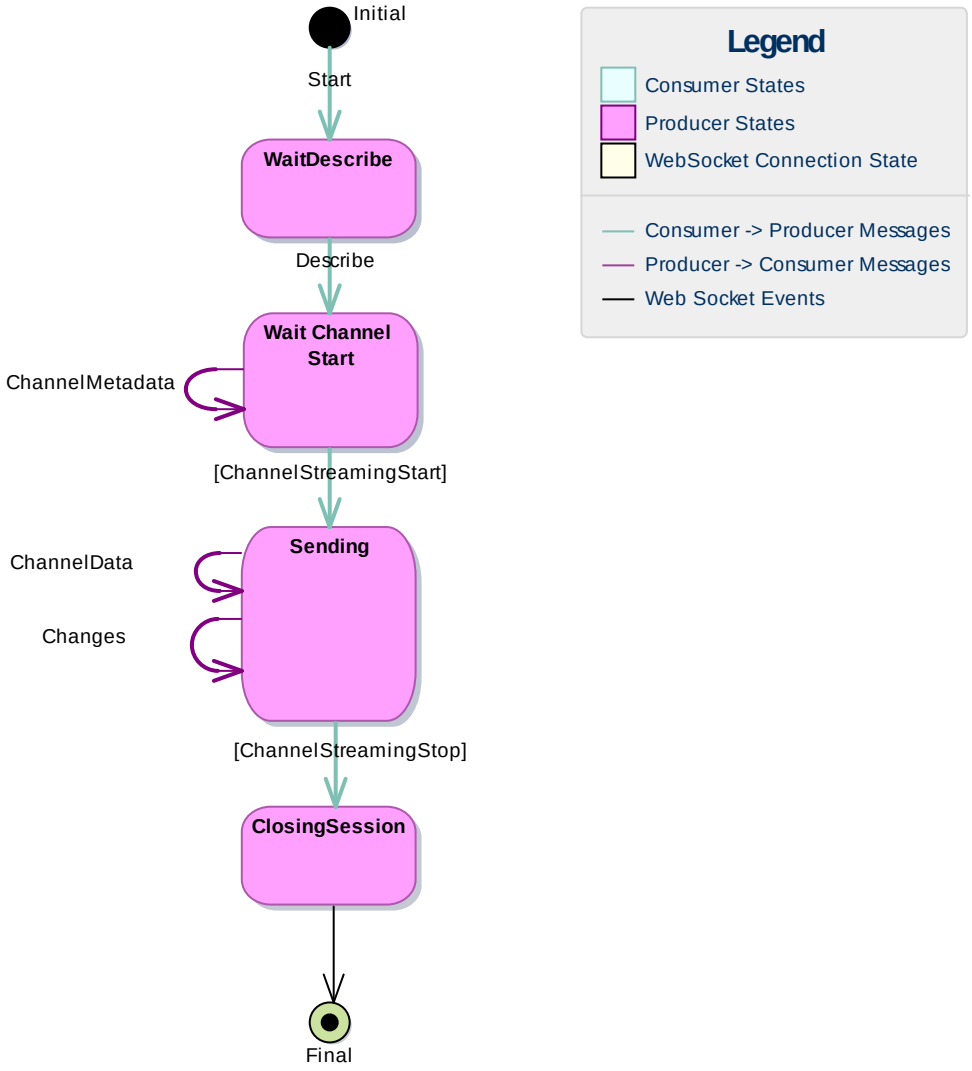

Figure 12: Channel Producer State

2.3 Discovery

The Discovery protocol uses a RESTful approach to enable store customers to enumerate and understand the contents of a store of data objects.

The roles for Discovery protocol are store and customer. The store represents a database or storage of object information, which the customer requests information about it. These are the same roles used in the Store, StoreNotification, and GrowingObject protocols.

This protocol has only two messages:

- One is sent from a customer to a store that supplies a URI and requests a list of child URIs.
- The other is sent from a store to a customer and contains the list of child URIs.

2.3.1 Interface: IDiscoveryStore

IDiscoveryStore describes the interface that MUST be implemented by the store role of the Discovery protocol.

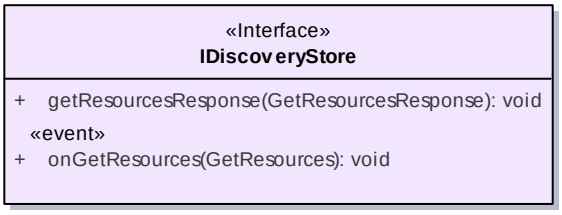

Figure 13: IDiscoveryStore

2.3.1.1 Methods

Method	Description	Type	Parameter Summary
onGetResources	A request to enumerate a URI. An ETP server MUST support a URI of "/".	event	(in) eventData: GetResources
getResourcesResponse	The asynchronous response message to the GetResource message; it contains the child URIs.		(in) eventData: GetResourcesResponse

2.3.2 Interface: IDiscoveryCustomer

IDiscoveryCustomer describes the interface that MUST be implemented by the customer role of the Discovery protocol.

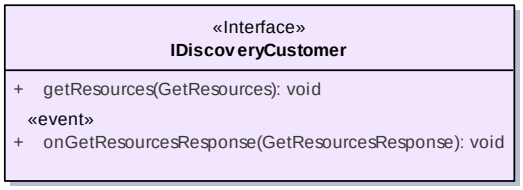

Figure 14: IDiscoveryCustomer

2.3.2.1 Methods

Method	Description	Type	Parameter Summary
getResources	Request for the server to enumerate child resources of a URI.		(in) eventData: GetResources
onGetResourcesResponse	Handles the response to a GetResources message.	event	(in) eventData: GetResourcesResponse

2.3.3 Sequence

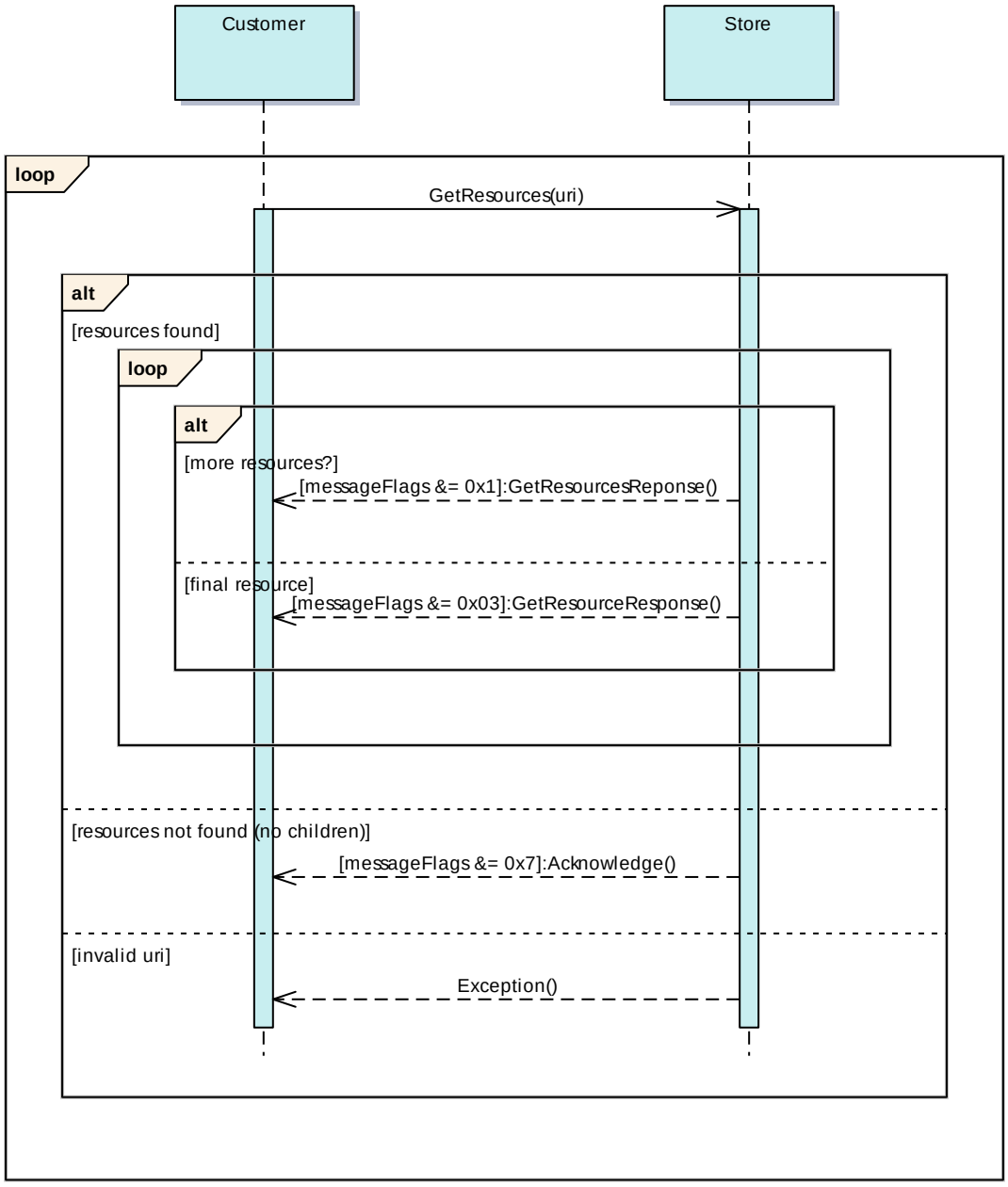

Figure 15: Discovery

2.4 Store

The Store protocol is used to perform CRUD operations (create, retrieve, update and delete) on data objects in a store. The roles for the Store protocol are store and customer.

The Store protocol uses 'Upsert' semantics for the create and update operations. That is, a single message 'PutObject' is used for both. If the object already exists, it is updated (i.e., completely replaced) by the new version. If the object does not exist, it is created. Unlike the WITSML SOAP protocols, there are no partial updates of objects and it is not possible to update individual fields in an object.

2.4.1 Requirements

Requirement	Type	Description
Supported Objects	Functional	A list of supported objects is defined by the store in the initial session/protocol negotiation (supportedObjects). For all messages in this protocol, the object MUST be supported by the store, either explicitly or by wild card. Use of an unsupported object by either role, MUST result in an EUNSUPPORTED_OBJECT exception.

2.4.2 Interface: IStoreStore

IStoreStore defines the interface that MUST be implemented by the store role of the Store protocol.

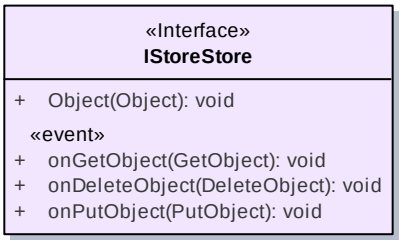

Figure 16: IStoreStore

2.4.2.1 Methods

Method	Description	Type	Parameter Summary
onGetObject	A request for the store to return a data object, identified by a URI.	event	(in) eventData: GetObject
onDeleteObject	A request to delete a data object from the store.	event	(in) eventData: DeleteObject
onPutObject	A request to insert or update a data object in the store.	event	(in) eventData: PutObject
Object	Return a data object from a GetObject request.		(in) eventData: Object

2.4.3 Interface: IStoreCustomer

IStoreCustomer defines the interface that MUST be implemented by the customer role of the Store protocol.

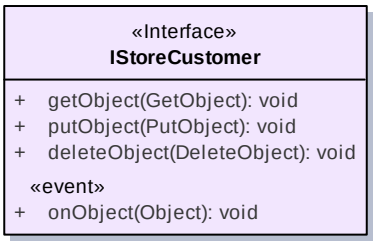

Figure 17: IStoreCustomer

2.4.3.1 Methods

Method	Description	Type	Parameter Summary
getObject	Get an object from the store, using a URI.		(in) eventData: GetObject
putObject	Put (create or update) an object in the store.		(in) eventData: PutObject
deleteObject	Delete an object from the store.		(in) eventData: DeleteObject
onObject	A data object returned from the GetObject message.	event	(in) eventData: Object

2.5 StoreNotification

The StoreNotification protocol is used allow store customers to receive notification of changes to data objects in the store in an event-driven manner. Customers subscribe to changes within a given context in the store (defined by a URI). The store provides notifications to the customer (while the session is valid) of additions, changes, and deletions from the store.

2.5.1 Interface: IStoreNotificationStore

IStoreNotificationStore describes the interface that must be implemented by the store role of the StoreNotification protocol.

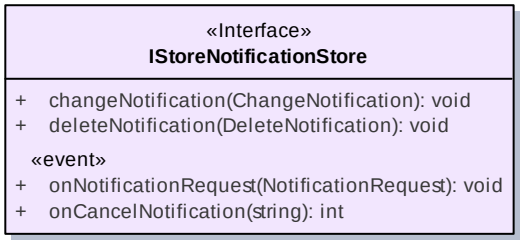

Figure 18: IStoreNotificationStore

2.5.1.1 Methods

Method	Description	Type	Parameter Summary
onNotificationRequest	A request from a customer to be notified when changes occur within the context of the supplied URI.	event	(in) eventData: NotificationRequest
changeNotification	Notification that an object has been created or changed within the context of a NotificationRequest URI.		(in) eventData: ChangeNotification
deleteNotification	Notification that an object has been deleted in the context of a subscription.		(in) eventData: DeleteNotification
onCancelNotification	Sent from a customer to a store to cancel a previous request for notification of change.	event	(in) eventData: string

2.5.2 Interface: IStoreNotificationCustomer

IStoreNotificationCustomer describes the interface that MUST be implemented by the customer role of the StoreNotification protocol.

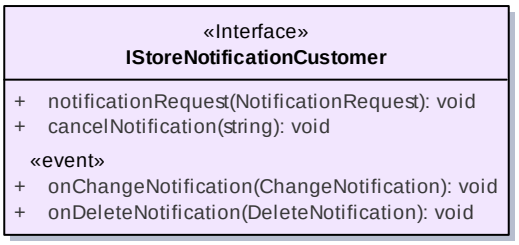

Figure 19: IStoreNotificationCustomer

2.5.2.1 Methods

Method	Description	Type	Parameter Summary
notificationRequest	A request from a customer to be notified when changes occur within the context of the supplied URI.		(in) eventData: NotificationRequest
onChangeNotification	Notification that an object has been created or changed within the context of a NotificationRequest URI.	event	(in) messageData: ChangeNotification
onDeleteNotification	Notification that an object has been deleted in the context of a subscription.	event	(in) eventData: DeleteNotification
cancelNotification	Sent from a customer to a store to cancel a previous request for notification of change.		(in) messageData: string

2.6 GrowingObject

The Growing Object protocol is used to manage the growing parts of data objects that are index-based (i.e., time and depth) but are not appropriate for the Channel Streaming protocol. For WITSML, these objects include:

- trajectory stations
- wellbore geology intervals
- wellbore geometry sections

The messages in this protocol are different from the Store protocol in that each of them is sent in the context of a parent object, and involves sending/receiving one or more XML fragments for the growing part of the object. Throughout this protocol documentation, the term "parent object" is used to refer to the main data object (i.e., trajectory, wellbore geology, etc.) that the list elements are attached to. In all messages, this object is always referenced by its UUID.

2.6.1 Interface: IGrowingObjectCustomer

IGrowingObjectCustomer describes the interface that MUST be implemented by the customer role of the GrowingObject protocol.

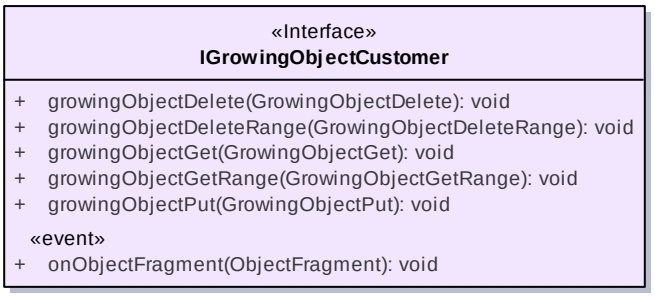

Figure 20: IGrowingObjectCustomer

2.6.1.1 Methods

Method	Description	Type	Parameter Summary
onObjectFragment	Contains a single list item. Used as the return message for Get and GetRange.	event	(in) eventData: ObjectFragment
growingObjectDelete	Delete one list item in a growing object.		(in) eventData: GrowingObjectDelete
growingObjectDeleteRange	Delete all list items in a range of index values.		(in) eventData: GrowingObjectDeleteRange
growingObjectGet	Get a single list item in a growing object, by its ID.		(in) eventData: GrowingObjectGet
growingObjectGetRange	Get all list items in a growing object within an index range.		(in) eventData: GrowingObjectGetRange
growingObjectPut	Add or update a list item in a growing object.		(in) eventData: GrowingObjectPut

2.6.2 Interface: IGrowingObjectStore

IGrowingObjectCustomer describes the interface that MUST be implemented by the store role of the GrowingObject protocol.

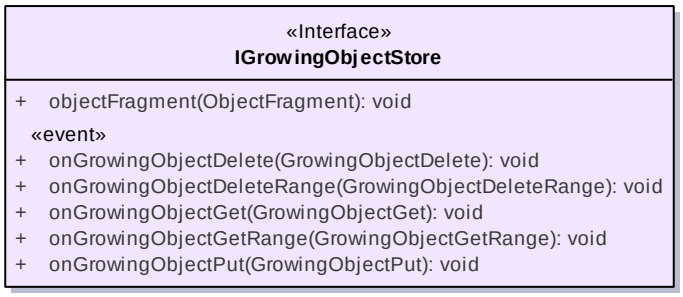

Figure 21: IGrowingObjectStore

2.6.2.1 Methods

Method	Description	Type	Parameter Summary
objectFragment	Contains a single list item. Used as the return message for Get and GetRange.		(in) eventData: ObjectFragment
onGrowingObjectDelete	Delete one list item in a growing object.	event	(in) eventData: GrowingObjectDelete
onGrowingObjectDeleteRange	Delete all list items in a range of index values.	event	(in) eventData: GrowingObjectDeleteRange
onGrowingObjectGet	Get a single list item in a growing object, by its ID.	event	(in) eventData: GrowingObjectGet
onGrowingObjectGetRange	Get all list items in a growing object within an index range.	event	(in) eventData: GrowingObjectGetRange
onGrowingObjectPut	Add or update a list item in a growing object.	event	(in) eventData: GrowingObjectPut

2.7 DataArray

The Data Array protocol is used to transfer large, binary arrays of heterogeneous data values, which Energistics domain standards typically store using HDF5. For example, RESQML uses HDF5 to store seismic, interpretation, and modeling data. As such, the arrays that are transferred with the DataArray protocol are logical versions of the HDF5 data sets.

2.7.1 Interface: IDataArrayStore

IDataArrayStore describes the interface that MUST be implemented by the store role of the DataArray protocol.

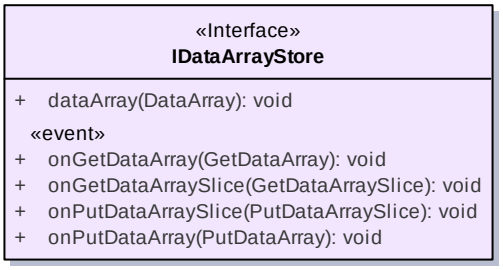

Figure 22: IDataArrayStore

2.7.1.1 Methods

Method	Description	Type	Parameter Summary
onGetDataArray	A request for a data array, referenced by URI.	event	(in) eventData: GetDataArray
onGetDataArraySlice	A request for a portion of a data array, referenced by URI and with an index range for each dimension.	event	(in) eventData: GetDataArraySlice
dataArray	The response to the GetDataArray and GetDataArraySlice message.		(in) eventData: DataArray
onPutDataArraySlice		event	(in) eventData: PutDataArraySlice
onPutDataArray		event	(in) eventData: PutDataArray

2.7.2 Interface: IDataArrayCustomer

IDataArrayCustomer describes the interface that MUST be implemented by the customer role of the DataArray protocol.

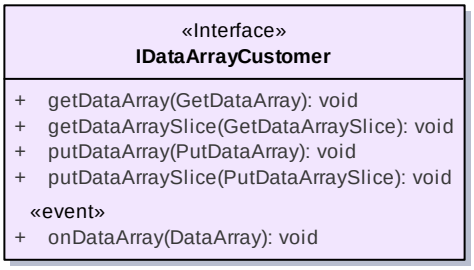

Figure 23: IDataArrayCustomer

2.7.2.1 Methods

Method	Description	Type	Parameter Summary
getDataArray	A request for a data array, referenced by URI.		(in) eventData: GetDataArray
getDataArraySlice	A request for a portion of a data array, referenced by URI and with an index range for each dimension.		(in) eventData: GetDataArraySlice
onDataArray	The response to the GetDataArray and GetDataArraySlice message.	event	(in) eventData: DataArray
putDataArray			(in) eventData: PutDataArray
putDataArraySlice			(in) eventData: PutDataArraySlice

3 Schemas

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 (ETP). These are defined using the following UML stereotypes and conventions:

- 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, second, schema authors must be careful to keep the ordering consistent between versions, to provide maximum interoperability.
- 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 example, the DateTime record is used to define how a date is transferred in all messages.
- 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 in ETP.
- Union: Used to represent a type that can be any one of a selected list of types. Each type is reflected in the UML as an attribute of the union class itself. Union more or less maps to the xsd:choice element in XML schemas.
- Map: UML does not support maps very well natively. However, in ETP, we can simplify because all Avro 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. These concepts are reflected in the Avro schema generation rules.

The Avro schemas, in JSON form, are produced automatically by the code-generation process in Enterprise Architect (EA). This 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 Avro Protocol (.avpr) file. Note that while the .avpr format is a convenient way to place all of the schemas in a single file, ETP DOES NOT use the Avro RPC protocol.

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.

3.1 Energistics

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

3.2 Datatypes

The datatypes package is intended to hold only low-level types that will be broadly re-used in various protocols. In general, primitive datatypes follow the rules for Avro itself.

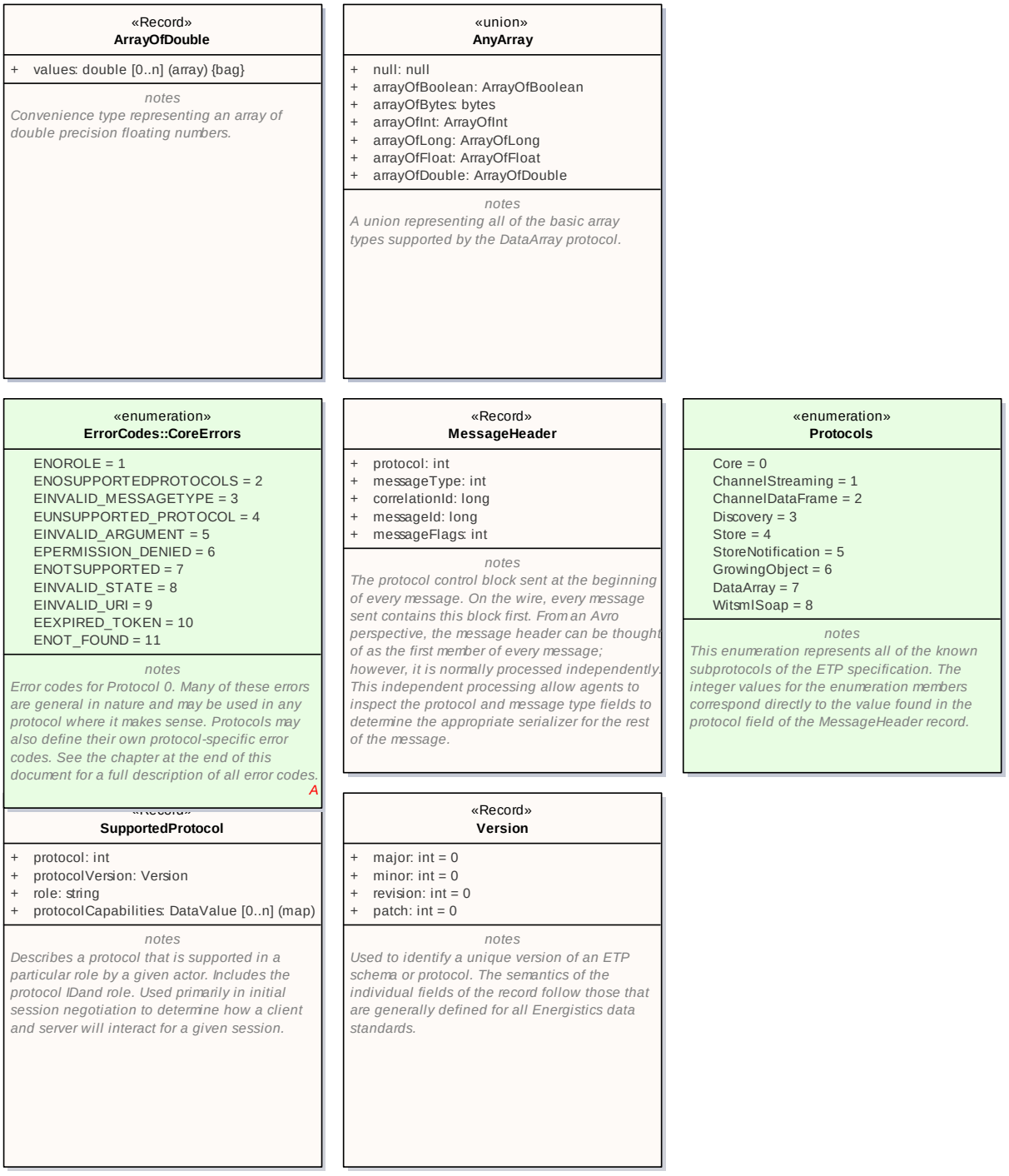

Figure 24: Datatypes

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

3.2.1 WebSocket Headers

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

- The **Sec-WebSocket-Protocol** header is defined in RFC-6455 and MUST be used to establish the use of the Energistics Transfer Protocol for this WebSocket connection. The value MUST be equal to the string "**energistics-tp**".
- 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 binary encoding.
- The client MAY supply a header named **etp-session**. The value of this header MUST be the UUID of a previously established session with this server. If the session stills exists on the server, then the server will reconnect the session according to the behavior described in the section on SessionSurvivability in the Core protocol.

Currently the Web browser version of the WebSocket API does not support passing any custom headers (the Sec-WebSocket-Protocol header is supported by the API, but nothing else). For this reason, 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, if present, use them. If both are present, the server SHOULD ignore the HTTP headers and use the query string.

3.2.2 Record: ServerCapabilities

Record describing the capabilities of a server. This record, though described in Avro, is not part of any ETP message. It simply describes the content of the JSON object that is used for pre-session server discovery.

Avro Schema

```
{
     "type": "record",
     "namespace": "Energistics.Datatypes",
     "name": "ServerCapabilities",
     "fields":
     [
         { "name": "applicationName", "type": "string" },
         { "name": "applicationVersion", "type": "string" },
         {
         "name": "supportedProtocols",
         "type": { "type": "array", "items": "Energistics.Datatypes.SupportedProtocol" }
     },
         {
         "name": "supportedObjects",
```

```
        "type": { "type": "array", "items": "string" }
    },
        { "name": "contactInformation", "type": "Energistics.Datatypes.Contact" },
        { "name": "supportedEncodings", "type": "string" }
    ]
}
```

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 as part of this string.	string	1	1
applicationVersion		string	1	1
supportedProtocols	An array of identifiers of the subprotocols supported by the server.	SupportedProtocol	1	n
supportedObjects	A list of objects that are supported for any of the object protocols (discovery, store, etc.) supported by this server. The form of the list is an array of strings, where each value is a contentType as described in the Energistics Identifier Specification. For example (serialized as JSON): "supportedObjects": ["application/x-witsml+xml;version=2.0;type=Well", "application/x-witsml+xml;version=2.0;type=Wellbore", "application/x-witsml+xml;version=2.0;type=Channel", "application/x-witsml+xml;version=2.0;type=Trajectory", "application/x-witsml+xml;version=2.0;type=part_TrajectoryStation", "application/x-prodml+xml;version=2.0;type=WellTest", "application/x-resqml+xml;version=2.0;type=TectonicBoundaryFeature"], Note that the contentType specifies both the schema family and version. If multiple versions are supported by the server, it includes a list of contentTypes for each version. As a special case, **type=*** can be used to specify that an entire schema release is supported. For example, the following means that this server supports the Well and Wellbore from either WITSML 1.4.1.1 or WITSML 2.0; the WellTest from PRODML 2.0 and all of RESQML 2.0.1. "supportedObjects": ["application/x-witsml+xml;version=1.4.1.1;type=Well", "application/x-witsml+xml;version=1.4.1.1;type=Wellbore", "application/x-witsml+xml;version=2.0;type=Well", "application/x-witsml+xml;version=2.0;type=Wellbore", "application/x-prodml+xml;version=2.0;type=WellTest", "application/x-resqml+xml;version=2.0.1;type=*"]	string	1	n
contactInformation	Contact information for this server.	Contact	1	1
supportedEncodings	Supported encodings for the server can be: • binary • JSON • binary;JSON	string	1	1

3.2.3 Protocols

This enumeration represents all of the known subprotocols of the ETP specification. The integer values for the enumeration members correspond directly to the value found in the protocol field of the MessageHeader record.

Enumeration	Value	Description
Core	0	The core protocol, it supports connection management, re-connect buffer, exception management, and message acknowledgement.
ChannelStreaming	1	The basic streaming protocol, it is used for simple devices (which can support only this protocol) and for publish/subscribe to send channel data.
ChannelDataFrame	2	The protocol used to retrieve a larger set of historical data from a channel provider, including channels that are no longer growing.
Discovery	3	The protocol used to navigate a data provider to find the objects, channels, etc., for which it can provide data.
Store	4	The protocol that handles CRUD operations on a store, similar to the functions of the WITSML V1.4.1 store interface.
StoreNotification	5	The protocol used to send notification of changes to data objects.
GrowingObject	6	The protocol used to manage the growing parts of data objects that are index-based (i.e., time and depth) but are not appropriate for the ChannelStreaming protocol.
DataArray	7	The protocol that transfers binary, homogeneous, multidimensional arrays of numbers.
WitsmlSoap	8	Wrapper around WITSML 1.x SOAP messages.

Avro Source

```
{
     "type": "enum",
     "namespace": "Energistics.Datatypes",
     "name": "Protocols",
     "symbols":
     [
         "Core",
         "ChannelStreaming",
         "ChannelDataFrame",
         "Discovery",
         "Store",
         "StoreNotification",
         "GrowingObject",
         "DataArray",
         "WitsmlSoap"
     ]
}
```

3.2.4 Record: Version

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	Involves significant change to all schemas, protocols, and business rules of a specification.	int	1	1
minor	Includes significant changes to schemas, most probably with breaking changes. The overall protocols and approach should not change significantly.	int	1	1
revision	May contain additions to existing schemas, but does not remove any schema elements. Enumerated types may also change.	int	1	1
patch	Involves minor changes only, usually bug fixes, and should not create breaking changes for other clients and servers on the same version.	int	1	1

3.2.5 Record: ArrayOfBoolean

Convenience type representing an array of Boolean values.

Avro Schema

```
{
     "type": "record",
     "namespace": "Energistics.Datatypes",
     "name": "ArrayOfBoolean",
     "fields":
     [
         {
         "name": "values",
```

```
        "type": { "type": "array", "items": "boolean" }
    }
    ]
}
```

Attribute	Description	Data Type	Min	Max
values	An array of Boolean values.	boolean	0	n

3.2.6 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

3.2.7 Record: ArrayOfInt

Convenience type representing an array of 4-byte integers.

Avro Schema

```
{
    "type": "record",
    "namespace": "Energistics.Datatypes",
    "name": "ArrayOfInt",
    "fields":
    [
        {
        "name": "values",
        "type": { "type": "array", "items": "int" }
    }
    ]
```

```
}
```

Attribute	Description	Data Type	Min	Max
values	An array of integers.	int	0	n

3.2.8 Record: ArrayOfFloat

Convenience type representing an array of 4-byte floats.

Avro Schema

```
{
     "type": "record",
     "namespace": "Energistics.Datatypes",
     "name": "ArrayOfFloat",
     "fields":
     [
         {
         "name": "values",
         "type": { "type": "array", "items": "float" }
     }
     ]
}
```

Attribute	Description	Data Type	Min	Max
values	An array of single precision numbers.	float	0	n

3.2.9 Record: ArrayOfLong

Convenience type representing an array of 8-byte long integers.

Avro Schema

```
{
     "type": "record",
     "namespace": "Energistics.Datatypes",
     "name": "ArrayOfLong",
     "fields":
     [
         {
         "name": "values",
         "type": { "type": "array", "items": "long" }
     }
     ]
}
```

Attribute	Description	Data Type	Min	Max
values	An array of long integers.	long	0	n

3.2.10 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

3.2.11 Record: MessageHeader

The protocol control block sent at the beginning of every message. On the wire, every message sent contains this block first. From an Avro perspective, the message header can be thought of as the first member of every message; however, it is normally processed independently. This independent processing 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	Identifies the protocol for which the message is intended. The values MUST come from the ProtocolID enumeration.	int	1	1
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 to use 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 example, an exception message MUST have as its correlationId the number of the message that caused the exception to be raised.	long	1	1
messageId	The unique identifier for this message within an ETP session.	long	1	1
messageFlags	A bit map of flags that apply to a message. It has the following bits defined: • 0x01: This message is a part of a multipart response. • 0x02: This message is the final part of a multi-message response. • 0x04: This message indicates that a potentially multi-message response has no data.	int	1	1

3.2.12 Record: SupportedProtocol

Describes a protocol that is supported in a particular role by a given actor. Includes the protocol IDand 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. Note that although the values will match the enumeration, it is not exhaustive and the value is specified as an integer, not an enumerated value. This usage is to allow private protocols to be negotiated.	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 can either be a "producer" or "consumer". The values expected for this string are defined by the subprotocols and included as decorations in the Avro schemas. Values are case-sensitive and, by modeling convention, should be all lower case in the specification and Avro schema files.	string	1	1
protocolCapabilities	A name-value map of protocol-specific configuration or capability data. The key names, defaults, optionality, and expected data types are defined as necessary by each protocol. Key names are always case-insensitive. Key names are strings, and key values are of type DataValue.	DataValue	0	n

3.2.13 Record: Contact

Contact information record for capabilities.

Avro Schema

```
{
    "type": "record",
    "namespace": "Energistics.Datatypes",
    "name": "Contact",
    "fields":
    [
        { "name": "organizationName", "type": ["null", "string"] },
        { "name": "contactName", "type": ["null", "string"] },
        { "name": "contactPhone", "type": ["null", "string"] },
        { "name": "contactEmail", "type": ["null", "string"] }
    ]
}
```

Attribute	Description	Data Type	Min	Max
organizationName		string	0	1
contactName		string	0	1
contactPhone		string	0	1
contactEmail		string	0	1

3.2.14 union: AnyArray

A union representing all of the basic array types supported by the DataArray protocol.

Attribute	Description	Data Type	Min	Max
null	Avro null.	null	1	1
arrayOfBoolean	Array of Boolean values.	ArrayOfBoolean	1	1
arrayOfBytes	Array of bytes.	bytes	1	1
arrayOfInt	Array of signed 32-bit integers	ArrayOfInt	1	1
arrayOfLong	Array of signed 64-bit integers.	ArrayOfLong	1	1
arrayOfFloat	Array of 4-byte floats.	ArrayOfFloat	1	1
arrayOfDouble	Array of 8-byte floats.	ArrayOfDouble	1	1

3.2.15 union: DataValue

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

Attribute	Description	Data Type	Min	Max
null	Avro null	null	1	1
double	Avro double	double	1	1
float	Avro float	float	1	1
int	Avro int	int	1	1
long	Avro long	long	1	1
string	Avro string	string	1	1
vector	Energistics.Datatypes.ArrayOfDouble	ArrayOfDouble	1	1
boolean	Avro Boolean	boolean	1	1
bytes		bytes	1	1

3.2.16 ChannelData

This schema package contains low-level types used to support both streaming (Protocol 1) and historical channel data (i.e., logs) (Protocol 2).

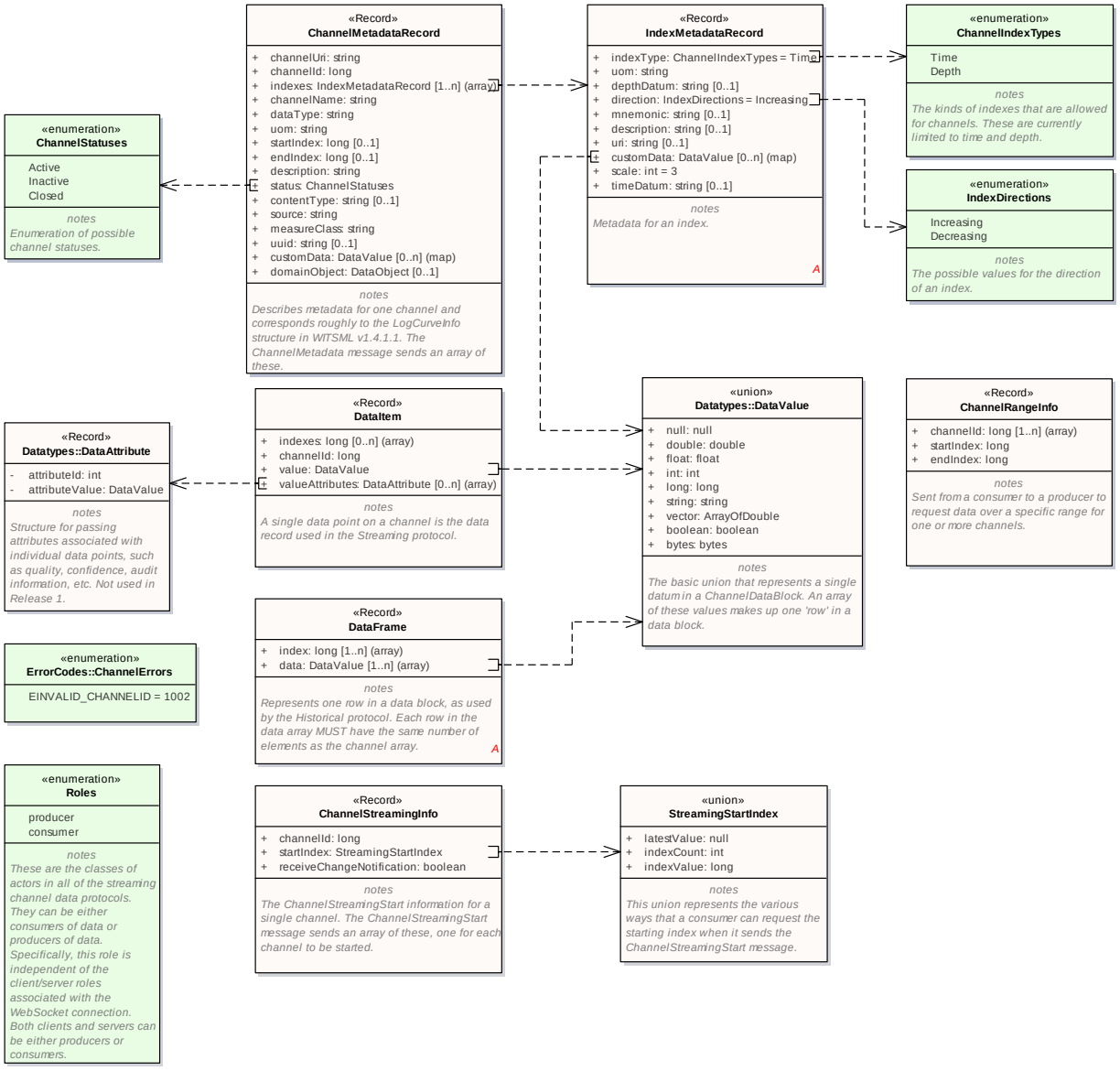

Figure 25: ChannelData - Common Types

3.2.16.1 Record: ChannelStreamingInfo

The ChannelStreamingStart information for a single channel. The ChannelStreamingStart message sends an array of these, one for each channel to be started.

Avro Schema

```
{
     "type": "record",
     "namespace": "Energistics.Datatypes.ChannelData",
     "name": "ChannelStreamingInfo",
     "fields":
     [
         { "name": "channelId", "type": "long" },
         { "name": "startIndex", "type": "Energistics.Datatypes.ChannelData.StreamingStartIndex"
},
         { "name": "receiveChangeNotification", "type": "boolean" }
     ]
}
```

Attribute	Description	Data Type	Min	Max
channelId	The ID of the channel to be started or stopped.	long	1	1
startIndex	The requested starting index for streaming.	StreamingStartIndex	1	1
receiveChangeNotification	Indicates that the consumer wants to receive ChannelDataChange messages (notification of non-real-time updates, and deletes on this channel) in addition to ChannelData (i.e., new real-time points).	boolean	1	1

3.2.16.2 ChannelStatuses

Enumeration of possible channel statuses.

Attribute	Description	Data Type	Min	Max
Active	Channel is currently producing. Same as ObjectGrowing = true in WITSML 1.x		1	1
Inactive	Channel is not currently producing data points. Same as ObjectGrowing = False in WITSML 1.x		1	1
Closed	Channel has 'gone away' and will never be used again.		1	1

Avro Source

```
{
     "type": "enum",
     "namespace": "Energistics.Datatypes.ChannelData",
     "name": "ChannelStatuses",
     "symbols":
     [
         "Active",
         "Inactive",
         "Closed"
     ]
```

```
}
```

3.2.16.3 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 WebSocket connection. Both clients and servers can be either producers or consumers.

Attribute	Description	Data Type	Min	Max
producer			1	1
consumer			1	1

Avro Source

```
{
     "type": "enum",
     "namespace": "Energistics.Datatypes.ChannelData",
     "name": "Roles",
     "symbols":
     [
         "producer",
         "consumer"
     ]
}
```

3.2.16.4 IndexDirections

The possible values for the direction of an index.

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

Avro Source

```
{
     "type": "enum",
     "namespace": "Energistics.Datatypes.ChannelData",
     "name": "IndexDirections",
     "symbols":
     [
         "Increasing",
         "Decreasing"
     ]
}
```

3.2.16.5 ChannelIndexTypes

The kinds of indexes that are allowed for channels. These are currently limited to time and depth.

Attribute	Description	Data Type	Min	Max
Time	Index is in time. The value is serialized in Avro as a long int, which represents microseconds from the time datum in IndexMetadataRecord.		1	1
Depth			1	1

Avro Source

```
{
     "type": "enum",
     "namespace": "Energistics.Datatypes.ChannelData",
     "name": "ChannelIndexTypes",
     "symbols":
     [
         "Time",
         "Depth"
     ]
}
```

3.2.16.6 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": "long" }
     },
         { "name": "channelId", "type": "long" },
         { "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 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.	long	0	n
channelId	The identifier of the channel for this point, as received in a ChannelMetadata record.	long	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 NOT specify any such value metadata, and this field is reserved for future releases.	DataAttribute	0	n

3.2.16.7 Record: DataFrame

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.ChannelData",
     "name": "DataFrame",
     "fields":
     [
         {
         "name": "index",
         "type": { "type": "array", "items": "long" }
     },
         {
         "name": "data",
         "type": { "type": "array", "items": "Energistics.Datatypes.DataValue" }
     }
     ]
}
```

Attribute	Description	Data Type	Min	Max
index		long	1	n
data		DataValue	1	n

3.2.16.8 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": "depthDatum", "type": ["null", "string"] },
         { "name": "direction", "type": "Energistics.Datatypes.ChannelData.IndexDirections" },
         { "name": "mnemonic", "type": ["null", "string"] },
         { "name": "description", "type": ["null", "string"] },
         { "name": "uri", "type": ["null", "string"] },
         { "name": "customData", "type": { "type": "map", "values":
"Energistics.Datatypes.DataValue" } },
         { "name": "scale", "type": "int" },
         { "name": "timeDatum", "type": ["null", "string"] }
     ]
}
```

Attribute	Description	Data Type	Min	Max
indexType	Main type of the index (time, depth, etc) as defined by ChannelIndexTypes enumeration.	ChannelIndexTypes	1	1
uom	The units of measure for the index. • For time or relative time indexes, this value is ignored, because 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. This is the UOM of the index AFTER scaling is applied to the long value in the index field of data points.	string	1	1
depthDatum	Describes the vertical datum for depth indexes. For WITSML channels, this MUST be supplied as a URI to the relevant well datum in the store associated with the channel. This can normally be a relative URI, in the form of a **uid** string for the referenced datum. For example, if we have a well with a datum with the uid 'KB', either of the following URIs would be correct: 'eml:///witsml14/well(1d8af245-8da9-4ece-86cd-e115e9c88a3e)/wellDatum(KB)'	string	0	1

Attribute	Description	Data Type	Min	Max
	'KB' The second form assumes that it is clear to the consumer what is the reference well for the channel involved. ETP itself, as a pure exchange protocol, has no knowledge of wells, etc. However, it normally is the case for a WITSML implementation that URIs of channels make clear the well context.			
direction	The direction of the index values, increasing or decreasing. Must remain constant for the life of a channel.	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. Optional field that allows an index to reference a permanent object on the producer or server.	string	0	1
customData	Key-value map of additional index information. Currently there are no standardized keys, and this is used only for vendor-specific information. Standard keys and values may be defined in the future.	DataValue	0	n
scale	For depth indexes only. Defines a power of 10 scale that is applied to the long integer value in the index field of all data points. For example, using a depth index whose UOM is feet, with a desired precision of hundredths of a foot, the scale is 2. The integer value 12345 represents a depth of 123.45 ft and a value of 12300 is used to represent exactly 123.00 ft. This value is used in all messages that reference a depth channel.	int	1	1
timeDatum	If the indexType is time, then this field can provide a datum for all time values in the channel. If supplied, it MUST be a UTC time in ISO 8601 string format (i.e., must have a trailing 'z' as the time zone. If it is null, then the datum is assumed to be the epoch of 12 AM Jan 1, 1970. The time of a given data point is determined by taking this datum in microseconds and adding the value of the index for that data point.	string	0	1

3.2.16.9 Record: ChannelMetadataRecord

Describes metadata for one channel and corresponds roughly to the LogCurveInfo structure in WITSML v1.4.1.1. 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": "long" },
        {
        "name": "indexes",
        "type": { "type": "array", "items":
"Energistics.Datatypes.ChannelData.IndexMetadataRecord" }
    },
        { "name": "channelName", "type": "string" },
        { "name": "dataType", "type": "string" },
        { "name": "uom", "type": "string" },
        { "name": "startIndex", "type": ["null", "long"] },
        { "name": "endIndex", "type": ["null", "long"] },
        { "name": "description", "type": "string" },
        { "name": "status", "type": "Energistics.Datatypes.ChannelData.ChannelStatuses" },
        { "name": "contentType", "type": ["null", "string"] },
        { "name": "source", "type": "string" },
        { "name": "measureClass", "type": "string" },
        { "name": "uuid", "type": ["null", "string"] },
        { "name": "customData", "type": { "type": "map", "values":
"Energistics.Datatypes.DataValue" } },
        { "name": "domainObject", "type": ["null", "Energistics.Datatypes.Object.DataObject"] }
    ]
}
```

Attribute	Description	Data Type	Min	Max
channelUri	A URI for the channel. It MUST be unique within the context of a server.	string	1	1
channelId	A producer-defined integer identifier for the channel. Channel IDs are only unique or meaningful within a given session. If you start a new session, the same channel URI may result in a different channel ID .	long	1	1
indexes	The metadata for the indexes associated with this channel. • The array MUST have a length of at least 1. • The first element in the array is considered the primary index. • The values of the primary index MUST be unique within the channel. • Secondary indexes MAY NOT be used in range queries. • IA value for secondary indexes is NOT required on every data point.	IndexMetadataRecord	1	n
channelName	The name for the channel. In WITSML v1.4.1, this corresponds directly to the mnemonic in LogCurveInfo.	string	1	1
dataType	The fully-qualified name of the Avro data type for the channel (i.e., double, float, int, etc.) To use logicalType, as defined by the Avro spec for dates, times, etc. See the notes under the customData field of ChannelMetadataRecord.	string	1	1
uom	The unit of measure for the channel. All DataItems and DataRows 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	long	0	1

Attribute	Description	Data Type	Min	Max
	channel.			
endIndex	The last (highest) recorded primary index value for the channel. For active channels, this value is only good at the time the metadata is sent.	long	0	1
description	Human-readable description of the channel. It maps to the Description field of the Citation element of the Channel XML object.	string	1	1
status	Current status of this channel; any changes to this status during a session results in ChannelStatusChange notification.	ChannelStatuses	1	1
contentType	• If the channel data points are structured objects (e.g., TrajectoryStation), then this string is the contentType of the data points, as defined in the *Energistics Identifier Specification* (e.g., "application/x-witsml+xml;version=2.0;type=part_TrajectoryStation"). • If this value is not null or empty, then the dataType field MUST be **bytes**.	string	0	1
source	Source of the data in the channel. For WITSML, this contains the contractor name.	string	1	1
measureClass	For WITSML, it is the PWLS class of the channel.	string	1	1
uuid	The UUID of the channel object in the store. For WITSML, this normally is the UUID of a channel. However, in the case of streaming complex objects over Protocol 1 (i.e., TrajectoryStation or WBGeometrySection) there is no corresponding channel object, and this is the UUID of the parent trajectory, etc.	string	0	1
customData	Name-Value pair (implemented as an Avro map) of custom values for the ChannelMetadata. This may contain both well-known (and thus, reserved) values as well as application and vendor-specific values. Names are case sensitive. The following are currently defined: **logicalType:** String Value. Used to define logical types in channel data, as permitted by the Avro specification (https://avro.apache.org/docs/1.8.0/spec.html#Logical+Types). Specifically, these values are supported: decimal, date, time-millis, time-micros, timestamp-millis, timestamp-micros, and duration. **precision:** Integer value. Used for the precision of a decimal logicalType. **scale** - Integer value. Used for the scale of a decimal logicalType.	DataValue	0	n
domainObject	Optional domain-specific data object describing the channel. This allows more complete channel metadata (such as a complete description of dimensions for an array channel) to be passed to the consumer. The field is optional for ETP. Individual ML implementations may require this field in certain circumstances.	DataObject	0	1

3.2.16.10 Record: ChannelRangeInfo

Sent from a consumer to a 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": { "type": "array", "items": "long" }
    },
        { "name": "startIndex", "type": "long" },
        { "name": "endIndex", "type": "long" }
    ]
}
```

Attribute	Description	Data Type	Min	Max
channelId	One or more channel IDs for which this range is requested. All channels MUST have a common index type, UOM, and direction. WITSML v1.4.1 servers are only required to support groupings of channels from the same log.	long	1	n
startIndex	The start of the data range.	long	1	1
endIndex	The end of the data range.	long	1	1

3.2.16.11 union: StreamingStartIndex

This union represents the various ways that a consumer can request the starting index when it sends the ChannelStreamingStart message.

Attribute	Description	Data Type	Min	Max
latestValue	Each channel in the subscription is streamed starting with its latest measured value.	null	1	1
indexCount	Begin streaming the channel from 'IndexCount' values before the latest value, inclusive. That is, if you specify a value of 10, you get the current value plus the 9 previous data points. If there are no IndexCount points in the channel history, you effectively get all of the channel points, but there are no notifications that less than IndexCount points are available.	int	1	1
indexValue	Begin streaming from the specified primary index value.	long	1	1

3.2.17 Object

The Object namespace contains datatypes for working with static objects. The datatypes in this package are used by the Discovery, Store, Query, and Store Notification protocols.

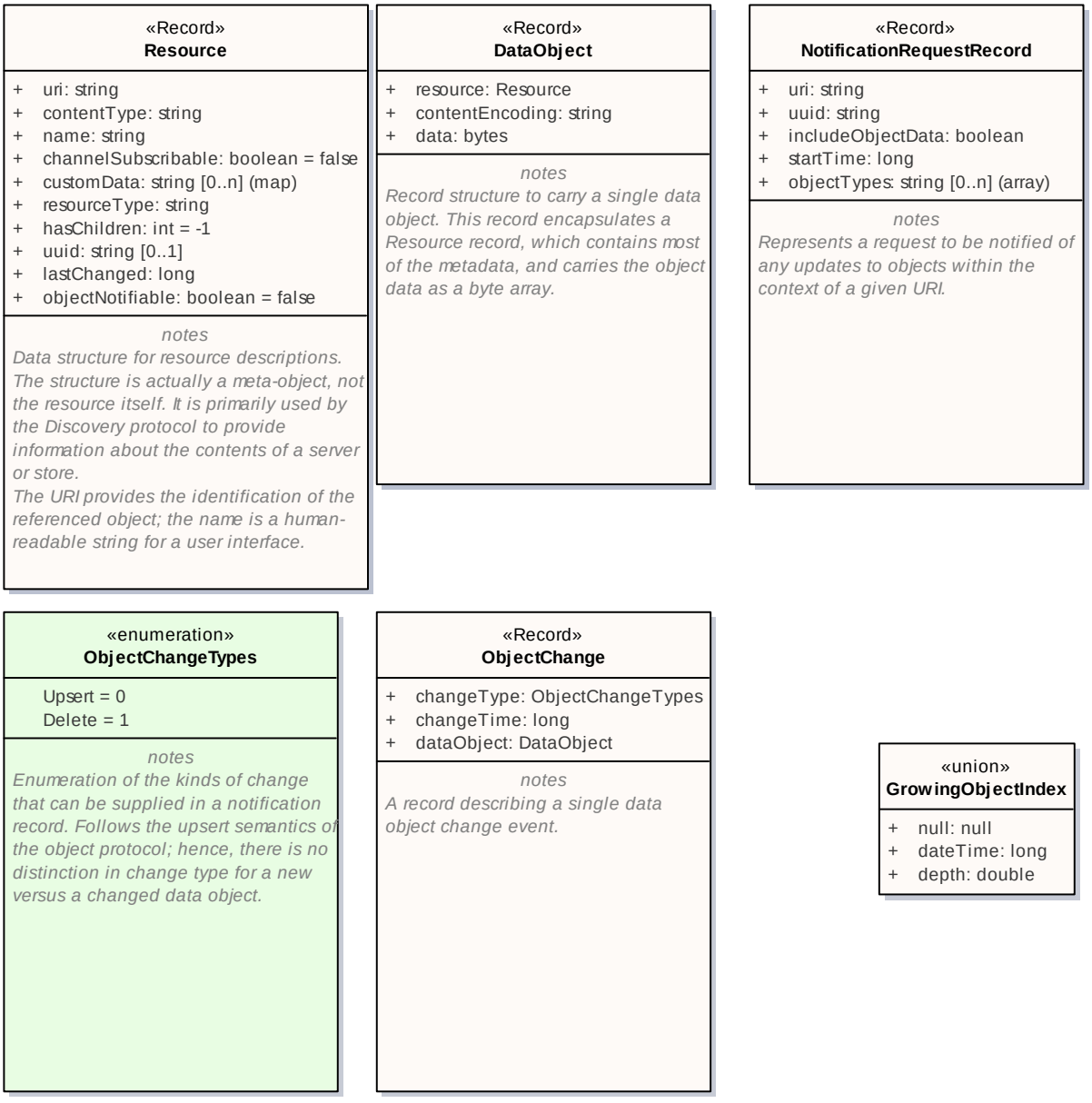

Figure 26: Object - Common Types

Common Types

3.2.17.1 Record: ObjectChange

A record describing a single data object change event.

Avro Schema

```
{
     "type": "record",
     "namespace": "Energistics.Datatypes.Object",
     "name": "ObjectChange",
     "fields":
     [
         { "name": "changeType", "type": "Energistics.Datatypes.Object.ObjectChangeTypes" },
         { "name": "changeTime", "type": "long" },
         { "name": "dataObject", "type": "Energistics.Datatypes.Object.DataObject" }
     ]
}
```

Attribute	Description	Data Type	Min	Max
changeType	The kind of change that occurred.	ObjectChangeTypes	1	1
changeTime	The time the data change event occurred. This is not the time the event happened, but the time that the change occurred in the store database.	long	1	1
dataObject	The object data. Based on the includeObjectData field of the NotificationRequestRecord, this MAY include only object metadata (i.e., the Resource information) or the full object data as XML.	DataObject	1	1

3.2.17.2 ObjectChangeTypes

Enumeration of the kinds of change that can be supplied in a notification record. Follows the upsert semantics of the object protocol; hence, there is no distinction in change type for a new versus a changed data object.

Attribute	Description	Data Type	Min	Max
Upsert	Object has been inserted or updated.	int	1	1
Delete	Object has been deleted	int	1	1

Avro Source

```
{
     "type": "enum",
     "namespace": "Energistics.Datatypes.Object",
     "name": "ObjectChangeTypes",
     "symbols":
     [
         "Upsert",
```

```
        "Delete"
    ]
}
```

3.2.17.3 Record: Resource

Data structure for resource descriptions. The structure is actually a meta-object, not the resource itself. It is primarily used by the Discovery protocol to provide information about the contents of a server or store.

The URI provides the identification of the referenced object; the name is a human-readable string for a user interface.

Avro Schema

```
{
    "type": "record",
    "namespace": "Energistics.Datatypes.Object",
    "name": "Resource",
    "fields":
    [
        { "name": "uri", "type": "string" },
        { "name": "contentType", "type": "string" },
        { "name": "name", "type": "string" },
        { "name": "channelSubscribable", "type": "boolean" },
        { "name": "customData", "type": { "type": "map", "values": "string" } },
        { "name": "resourceType", "type": "string" },
        { "name": "hasChildren", "type": "int" },
        { "name": "uuid", "type": ["null", "string"] },
        { "name": "lastChanged", "type": "long" },
        { "name": "objectNotifiable", "type": "boolean" }
    ]
}
```

Attribute	Description	Data Type	Min	Max
uri	A URI for the referenced object. It is understood that the supplier of this URI can de-reference it as a data object, or as some other kind of resource irrespective of the string contents of the URI. Specifically: • If the resource points to channels (i.e., it is a channel or the parent of many channels), then this URI could be passed to ChannelDescribe to get more information about those channels. • If the resource points to a static data object (i.e., resourceType == DataObject), then it could be passed to the Store protocol to get/put or delete that object.	string	1	1
contentType	Provides the application with the information needed to deserialize the referenced resource. In the case of a data object, this field is a string that is identical to the ContentType in an EPC file (see the *Energistics Packaging Conventions Specification*).	string	1	1

Attribute	Description	Data Type	Min	Max
	For example: "application/x-resqml+xml;version=2.0;type=TectonicBoundaryFeature" "application/x-witsml+xml;version=2.0;type=part_TrajectoryStation" This field is case insensitive.			
name	A human-readable name for the object. There is no expectation of uniqueness or any particular semantic for this value.	string	1	1
channelSubscribable	Indicates that this resource is the source of one or more channels for Protocol 1. Specifically, the URI in this record can be sent in the ChannelDescribe message with an expectation of receiving one or more channels.	boolean	1	1
customData	A key-value map of custom data about the resource. Both the key and the values are strings. Currently, there are no defined semantics for either key names or values.	string	0	n

resourceType	A level above Energistics data objects. Because discovery contains meta information about stores, it returns resource records that don't refer specifically to static data objects in the store. The following are defined as supported values for the resourceType field (the strings are case insensitive):
	
	• DataObject: Refers to a single data object in one of the Energistics specifications. It can thus be retrieved/altered or deleted by the Object protocol.
	• UriProtocol: The root for a well-known URI procotol. The details of any given URI protocol are generally part of a data specification and are outside the scope of ETP itself. For example, if an agent passes the string "/" to the Discovery protocol, the response might contain something like this:
	
	{
	uri: "eml://witsml20",
	resourceType: "UriProtocol"
	name: "WITSML 2.0 Store"
	}
	
	• Folder: The returned node is a container for other nodes. If the resource is of type Folder and the contentType field is non-null, then the folder's direct children MUST only be of that contentType, or other folders. For example, the URI "eml://witsml20" might return something like the following, which indicates that all of the children of this node are of type well data objects or other folders, such as wellbore:
	
	{
	resourceType: "Folder"
	name: "Wells"
	uri: "eml://witsml20/well"
	contentType: "application/x-witsml+xml;version=2.0;type=Well"
	}
		string	1	1

hasChildren	Indicates that the node has children. The semantic of "may have children" is not conveyed here, but it is intrinsic in the definition of the node types. For example, if the node is a well, an application would be expected to understand that it potentially has children that are wellbores. This value is one of the following: • -1 (child count is unknown)	int	1	1

Attribute	Description	Data Type	Min	Max
	• 0 (no children) • a positive integer (the count of children)			
uuid	When the resource is an Energistics data object, then this is the UUID of the object. This must be a UUID as specified by RFC 4122 (https://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	0	1
lastChanged	The date and time of the last change to this object, on the supplying store/provider.	long	1	1
objectNotifiable	Indicates that this resource's URI can be used as the input to the StoreNotification NotificationRequest message.	boolean	1	1

3.2.17.4 Record: DataObject

Record structure to carry a single data object. This record encapsulates a Resource record, which contains most of the metadata, and carries the object data as a byte array.

Avro Schema

```
{
     "type": "record",
     "namespace": "Energistics.Datatypes.Object",
     "name": "DataObject",
     "fields":
     [
         { "name": "resource", "type": "Energistics.Datatypes.Object.Resource" },
         { "name": "contentEncoding", "type": "string" },
         { "name": "data", "type": "bytes" }
     ]
}
```

Attribute	Description	Data Type	Min	Max
resource	Contains high-level metadata about the data object being transferred, in the form of a Resource struct. If you have used the Discovery protocol to find a Resource, then the result of the GetObject message should be to return the same resource information in this field.	Resource	1	1
contentEncoding	Indicates if compression is used. Currently, it 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. Note, for StoreNotification messages, if includeObjectData is false in the NotificationRequest, this field has zero bytes. For all 1.x Energistics schema data objects that use the plural root, the XML document MUST contain only a single object.	bytes	1	1

3.2.17.5 Record: NotificationRequestRecord

Represents a request to be notified of any updates to objects within the context of a given URI.

Avro Schema

```
{
     "type": "record",
     "namespace": "Energistics.Datatypes.Object",
     "name": "NotificationRequestRecord",
     "fields":
     [
         { "name": "uri", "type": "string" },
         { "name": "uuid", "type": "string" },
         { "name": "includeObjectData", "type": "boolean" },
         { "name": "startTime", "type": "long" },
         {
         "name": "objectTypes",
         "type": { "type": "array", "items": "string" }
     }
     ]
}
```

Attribute	Description	Data Type	Min	Max
uri	The content URI for the subscription. The exact definitions for allowable URIs are specified in the Energistics Identifier Specification.	string	1	1
uuid	A UUID for this notification request. This MUST be a newly-generated UUID from the customer sending the message. This ID can be used to remove the notification at a later point. This MUST be a UUID as specified by RFC 4122 (https://www.ietf.org/rfc/rfc4122.txt). The UUID MUST be formatted as a string, in the form commonly known as "Windows Registry format". Example: **65CCC68E-C0B4-454E-8757-3284F94AE861**	string	1	1
includeObjectData	• If true, then notification MUST contain the complete data object, corresponding to the change. • If false, only a Resource record for the object is sent and the Store protocol MUST be used to get the object itself. Growing parts of objects, such as trajectory stations or data arrays, MUST NOT be sent as a result of this parameter being set to 'true'.	boolean	1	1
startTime	Starting time for changes. If any of the objects within the context of the URI have changed since this time, then the store MUST immediately send one ChangeNotification message of the object in its current state. Note, the startTime is assumed to be in the past,	long	1	1

Attribute	Description	Data Type	Min	Max
	and is ignored if it is in the future (i.e., notifications are still sent, but the immediate sending of one message does not occur).			
objectTypes	An array of data object types for which change notification is requested. An empty array indicates all object types within the context of the supplied URI. The format of the object type string is exactly the same as the contentType field of the Resource structure.	string	0	n

3.2.17.6 union: GrowingObjectIndex

Attribute	Description	Data Type	Min	Max
null	null	null	1	1
dateTime	A UTC dateTime value, serialized as a long, using the Avro logical type **timestamp-micros** (microseconds from the Unix Epoch, 1 January 1970 00:00:00.000000 UTC).	long	1	1
depth	Depth value. The UOM and datum are supplied in the Message context.	double	1	1

3.3 Protocol

This package is the root namespace for all messages in all subprotocols of ETP. Each child package represents a protocol and is decorated with the following UML tagged values:

- ProtocolID. The assigned integer value for this protocol.
- Roles. A comma-delimited list of the role names used in the protocol. Roles identify the parties that send and/or receive specific messages in a protocol.

Each class in this section has 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)
- CorrelationId 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 and indicates that the messageFlags (bits 0x1 and 0x2) MUST be used on this message.
- SenderRole. A string defining the role that sends this message, as opposed to receive it. This allows implementers (human and machine) to know whether the interface should implement as an event or sent message. A value of '*' indicates that all roles may both send and receive the message.

3.3.1 Core

Stability: 3 - Stable

The Core protocol creates and manages an ETP session.

ProtocolID: 0

Defined Roles: client,server

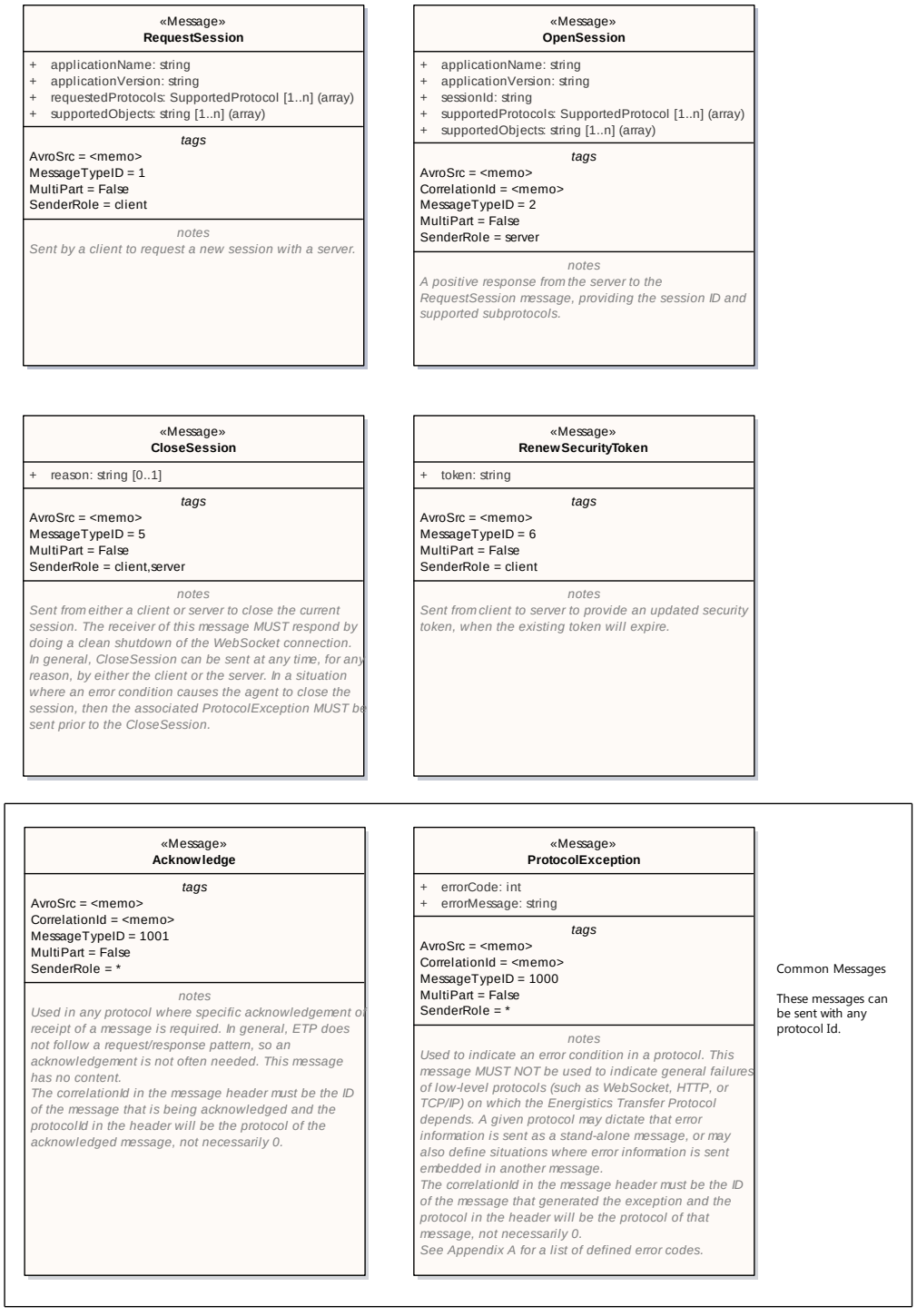

Figure 27: Core

3.3.1.1 Message: RequestSession

Sent by a client to request a new session with a server.

Message Type ID: 1

Correlation Id Usage: n/a

Multi-part: False

Sent by: client

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)

Attribute	Description	Data Type	Min	Max
applicationName	The string by which the client identifies itself, normally a software product or system name. This string may or may not include a version, and the format is entirely application dependent. Vendors are encouraged to identify their company name as part of this string.	string	1	1
applicationVersion		string	1	1
requestedProtocols	An array of protocol IDs that 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 that are supported.	SupportedProtocol	1	n
supportedObjects	A list of the Data Objects supported by the client. This list MUST be empty if the client is a customer. This field MUST be supplied if the client is a Store and is requesting a customer role for the server. For details on the content, see the documentation for OpenSession.	string	1	n

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.Core",
     "name": "RequestSession",
     "messageType": "1",
     "protocol": "0",
     "senderRole": "client",
     "protocolRoles": "client,server",
     "fields":
     [
         { "name": "applicationName", "type": "string" },
         { "name": "applicationVersion", "type": "string" },
         {
         "name": "requestedProtocols",
         "type": { "type": "array", "items": "Energistics.Datatypes.SupportedProtocol" }
```

```
    },
        {
        "name": "supportedObjects",
        "type": { "type": "array", "items": "string" }
    }
    ]
}
```

3.3.1.2 Message: OpenSession

A positive response from the server to the RequestSession message, providing the session ID and supported subprotocols.

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

Sent by: server

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 as part of this string.	string	1	1
applicationVersion		string	1	1
sessionId	An identifier for this session. This must be a UUID as specified by RFC 4122. 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 SupportedProtocol, representing the requested protocols that are actually supported by the server.	SupportedProtocol	1	n
supportedObjects	See the documentation for ServerCapabilities record for a description of this field.	string	1	n

Avro Source

```
{
    "type": "record",
    "namespace": "Energistics.Protocol.Core",
    "name": "OpenSession",
```

```
    "messageType": "2",
    "protocol": "0",
    "senderRole": "server",
    "protocolRoles": "client,server",
    "fields":
    [
        { "name": "applicationName", "type": "string" },
        { "name": "applicationVersion", "type": "string" },
        { "name": "sessionId", "type": "string" },
        {
        "name": "supportedProtocols",
        "type": { "type": "array", "items": "Energistics.Datatypes.SupportedProtocol" }
    },
        {
        "name": "supportedObjects",
        "type": { "type": "array", "items": "string" }
    }
    ]
}
```

3.3.1.3 Message: CloseSession

Sent from either a client or server to close the current session. The receiver of this message MUST respond by doing a clean shutdown of the WebSocket connection.

In general, CloseSession can be sent at any time, for any reason, by either the client or the server. In a situation where an error condition causes the agent to close the session, then the associated ProtocolException MUST be sent prior to the CloseSession.

Message Type ID: 5

Correlation Id Usage: n/a

Multi-part: False

Sent by: client,server

Attribute	Description	Data Type	Min	Max
reason	The reason for requesting the session be closed.	string	0	1

Avro Source

```
{
    "type": "record",
    "namespace": "Energistics.Protocol.Core",
    "name": "CloseSession",
```

```
    "messageType": "5",
    "protocol": "0",
    "senderRole": "client,server",
    "protocolRoles": "client,server",
    "fields":
    [
        { "name": "reason", "type": ["null", "string"] }
    ]
}
```

3.3.1.4 Message: ProtocolException

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 WebSocket, HTTP, or TCP/IP) on which the Energistics Transfer Protocol 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.

The correlationId in the message header must be the ID of the message that generated the exception and the protocol in the header will be the protocol of that message, not necessarily 0.

See Appendix A for a list of defined error codes.

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

Sent by: *

Attribute	Description	Data Type	Min	Max
errorCode	Unique number, defined by this specification, for an error. Error numbers are defined separately for each subprotocol, thus to unambiguously identify an error message, you must combine the ProtocolID and the errorCode.	int	1	1
errorMessage	A string representation of the error message.	string	1	1

Avro Source

```
{
    "type": "record",
    "namespace": "Energistics.Protocol.Core",
    "name": "ProtocolException",
    "messageType": "1000",
    "protocol": "0",
```

```
    "senderRole": "*",
    "protocolRoles": "client,server",
    "fields":
    [
        { "name": "errorCode", "type": "int" },
        { "name": "errorMessage", "type": "string" }
    ]
}
```

3.3.1.5 Message: Acknowledge

Used in any protocol where specific acknowledgement of receipt of a message is required. In general, ETP does not follow a request/response pattern, so an acknowledgement is not often needed. This message has no content.

The correlationId in the message header must be the ID of the message that is being acknowledged and the protocolId in the header will be the protocol of the acknowledged message, not necessarily 0.

Message Type ID: 1001

Correlation Id Usage:

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

Multi-part: False

Sent by: *

Avro Source

```
{
    "type": "record",
    "namespace": "Energistics.Protocol.Core",
    "name": "Acknowledge",
    "messageType": "1001",
    "protocol": "0",
    "senderRole": "*",
    "protocolRoles": "client,server",
    "fields":
    [

    ]
}
```

3.3.1.6 Message: RenewSecurityToken

Sent from client to server to provide an updated security token, when the existing token will expire.

Message Type ID: 6

Correlation Id Usage: n/a

Multi-part: False

Sent by: client

Attribute	Description	Data Type	Min	Max
token	The security token, exactly in the form that it provided in the authorization header.	string	1	1

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.Core",
     "name": "RenewSecurityToken",
     "messageType": "6",
     "protocol": "0",
     "senderRole": "client",
     "protocolRoles": "client,server",
     "fields":
     [
         { "name": "token", "type": "string" }
     ]
}
```

3.3.2 ChannelStreaming

Stability: 3 - Stable

The ChannelStreaming protocol supports the streaming of channel data.

ProtocolID: 1

Defined Roles: producer,consumer

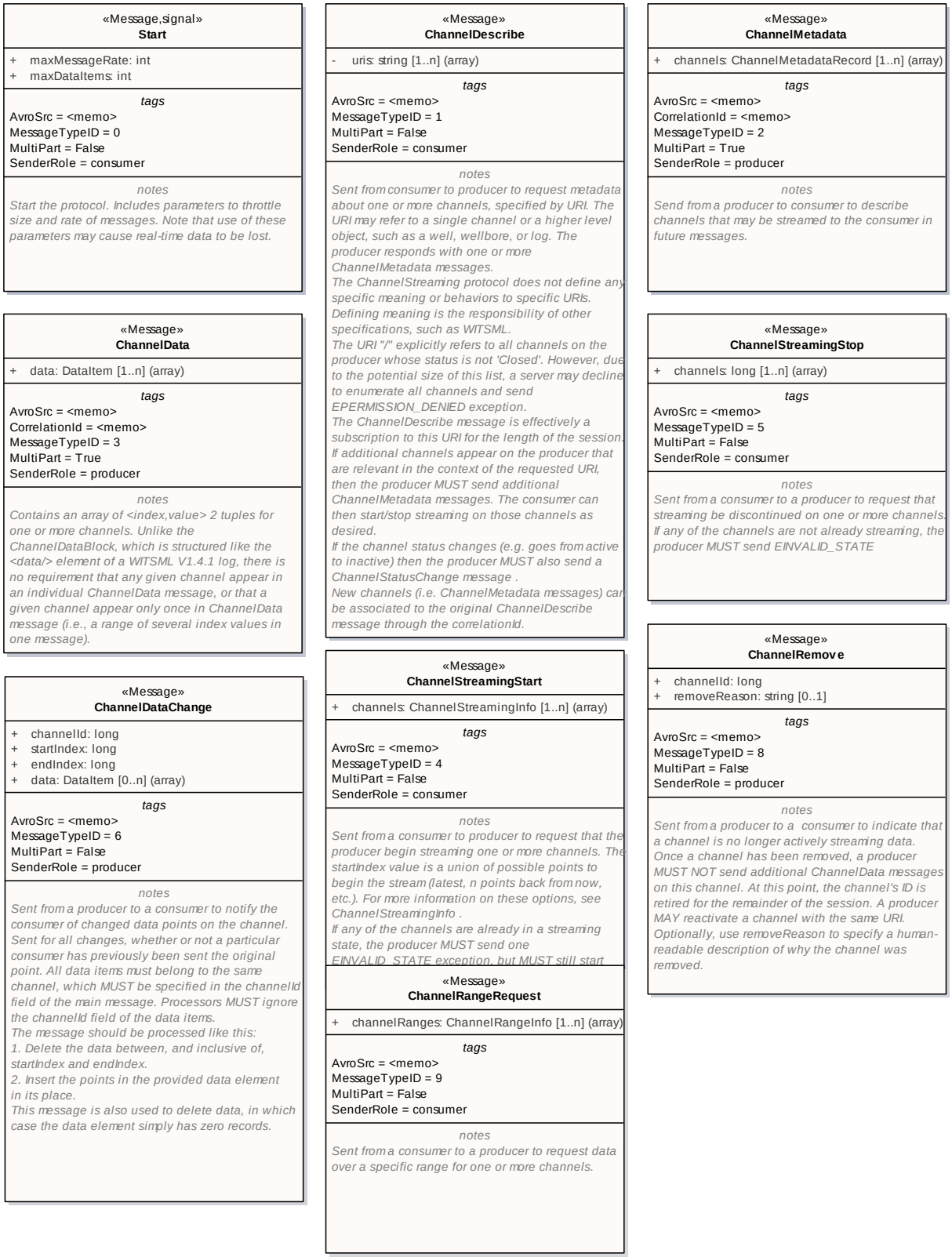

Figure 28: ChannelStreaming

3.3.2.1 Message: Start

Start the protocol. Includes parameters to throttle size and rate of messages. Note that use of these parameters may cause real-time data to be lost.

Message Type ID: 0

Correlation Id Usage: n/a

Multi-part: False

Sent by: consumer

Attribute	Description	Data Type	Min	Max
maxMessageRate	Do not send two channel messages within a smaller window than this value (in milliseconds).	int	1	1
maxDataItems	Do not send more than this number of data points in a single message.	int	1	1

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.ChannelStreaming",
     "name": "Start",
     "messageType": "0",
     "protocol": "1",
     "senderRole": "consumer",
     "protocolRoles": "producer,consumer",
     "fields":
     [
         { "name": "maxMessageRate", "type": "int" },
         { "name": "maxDataItems", "type": "int" }
     ]
}
```

3.3.2.2 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. The producer responds with one or more ChannelMetadata messages.

The ChannelStreaming protocol does not define any specific meaning or behaviors to specific URIs. Defining meaning is the responsibility of other specifications, such as WITSML.

The URI "/" explicitly refers to all channels on the producer whose status is not 'Closed'. However, due to the potential size of this list, a server may decline to enumerate all channels and send EPERMISSION_DENIED exception.

The ChannelDescribe message is effectively a subscription to this URI for the length of the session. If additional channels appear on the producer that are relevant in the context of the requested URI, then the producer MUST send additional ChannelMetadata messages. The consumer can then start/stop streaming on those channels as desired.

If the channel status changes (e.g. goes from active to inactive) then the producer MUST also send a ChannelStatusChange message .

New channels (i.e. ChannelMetadata messages) can be associated to the original ChannelDescribe message through the correlationId.

Message Type ID: 1

Correlation Id Usage: n/a

Multi-part: False

Sent by: consumer

Attribute	Description	Data Type	Min	Max
uris	A list of URIs to describe.	string	1	n

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.ChannelStreaming",
     "name": "ChannelDescribe",
     "messageType": "1",
     "protocol": "1",
     "senderRole": "consumer",
     "protocolRoles": "producer,consumer",
     "fields":
     [
         {
         "name": "uris",
         "type": { "type": "array", "items": "string" }
     }
     ]
}
```

3.3.2.3 Message: ChannelMetadata

Send from a producer to consumer to describe channels that 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

Sent by: producer

Attribute	Description	Data Type	Min	Max
channels	The list of channels.	ChannelMetadataRecord	1	n

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.ChannelStreaming",
     "name": "ChannelMetadata",
     "messageType": "2",
     "protocol": "1",
     "senderRole": "producer",
     "protocolRoles": "producer,consumer",
     "fields":
     [
         {
         "name": "channels",
         "type": { "type": "array", "items":
"Energistics.Datatypes.ChannelData.ChannelMetadataRecord" }
     }
     ]
}
```

3.3.2.4 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 V1.4.1 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

Sent by: producer

Attribute	Description	Data Type	Min	Max
data	Contains the channel data.	DataItem	1	n

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.ChannelStreaming",
     "name": "ChannelData",
     "messageType": "3",
     "protocol": "1",
     "senderRole": "producer",
     "protocolRoles": "producer,consumer",
     "fields":
     [
         {
         "name": "data",
         "type": { "type": "array", "items": "Energistics.Datatypes.ChannelData.DataItem" }
     }
     ]
}
```

3.3.2.5 Message: ChannelStreamingStart

Sent from a consumer to producer to request that the producer begin streaming one or more channels. The startIndex value is a union of possible points to begin the stream (latest, *n* points back from now, etc.). For more information on these options, see ChannelStreamingInfo .

If any of the channels are already in a streaming state, the producer MUST send one EINVALID_STATE exception, but MUST still start streaming all of the channels that are valid.

Message Type ID: 4

Correlation Id Usage: n/a

Multi-part: False

Sent by: consumer

Attribute	Description	Data Type	Min	Max
channels	An array of the channels to start streaming.	ChannelStreamingInfo	1	n

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.ChannelStreaming",
     "name": "ChannelStreamingStart",
```

```
    "messageType": "4",
    "protocol": "1",
    "senderRole": "consumer",
    "protocolRoles": "producer,consumer",
    "fields":
    [
        {
        "name": "channels",
        "type": { "type": "array", "items":
"Energistics.Datatypes.ChannelData.ChannelStreamingInfo" }
    }
    ]
}
```

3.3.2.6 Message: ChannelStreamingStop

Sent from a consumer to a producer to request that streaming be discontinued on one or more channels. If any of the channels are not already streaming, the producer MUST send EINVALID_STATE

Message Type ID: 5

Correlation Id Usage: n/a

Multi-part: False

Sent by: consumer

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

Avro Source

```
{
    "type": "record",
    "namespace": "Energistics.Protocol.ChannelStreaming",
    "name": "ChannelStreamingStop",
    "messageType": "5",
    "protocol": "1",
    "senderRole": "consumer",
    "protocolRoles": "producer,consumer",
    "fields":
    [
        {
        "name": "channels",
        "type": { "type": "array", "items": "long" }
    }
```

```
    ]
}
```

3.3.2.7 Message: ChannelDataChange

Sent from a producer to a consumer to notify the consumer of changed data points on the channel. Sent for all changes, whether or not a 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 like this:

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

This message is also used to delete data, in which case the data element simply has zero records.

Message Type ID: 6

Correlation Id Usage: n/a

Multi-part: False

Sent by: producer

Attribute	Description	Data Type	Min	Max
channelId	The channel to which all changes apply. The channelId in all data records MUST be the same as this value.	long	1	1
startIndex	The starting index of the data to be replaced by the supplied data.	long	1	1
endIndex	The ending index of the range to be replaced by the supplied data.	long	1	1
data	An array of data items to be replaced. To delete all points in the range, send an empty array.	DataItem	0	n

Avro Source

```
{
    "type": "record",
    "namespace": "Energistics.Protocol.ChannelStreaming",
    "name": "ChannelDataChange",
    "messageType": "6",
    "protocol": "1",
    "senderRole": "producer",
    "protocolRoles": "producer,consumer",
    "fields":
    [
        { "name": "channelId", "type": "long" },
```

```
        { "name": "startIndex", "type": "long" },
        { "name": "endIndex", "type": "long" },
        {
        "name": "data",
        "type": { "type": "array", "items": "Energistics.Datatypes.ChannelData.DataItem" }
    }
    ]
}
```

3.3.2.8 Message: ChannelRemove

Sent from a producer to a consumer to indicate that a channel is no longer actively streaming data. Once a channel has been removed, a producer MUST NOT send additional ChannelData messages on this channel. At this point, the channel's ID is retired for the remainder of the session. A producer MAY reactivate a channel with the same URI.

Optionally, use removeReason to specify a human-readable description of why the channel was removed.

Message Type ID: 8

Correlation Id Usage: n/a

Multi-part: False

Sent by: producer

Attribute	Description	Data Type	Min	Max
channelId	The ID of the channel to be deleted.	long	1	1
removeReason	An optional human-readable description of why the channel is being removed.	string	0	1

Avro Source

```
{
    "type": "record",
    "namespace": "Energistics.Protocol.ChannelStreaming",
    "name": "ChannelRemove",
    "messageType": "8",
    "protocol": "1",
    "senderRole": "producer",
    "protocolRoles": "producer,consumer",
    "fields":
    [
        { "name": "channelId", "type": "long" },
        { "name": "removeReason", "type": ["null", "string"] }
    ]
}
```

3.3.2.9 Message: ChannelRangeRequest

Sent from a consumer to a 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

Sent by: consumer

Attribute	Description	Data Type	Min	Max
channelRanges	An array of data that specifies the channels and the range of each for which data is being requested.	ChannelRangeInfo	1	n

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.ChannelStreaming",
     "name": "ChannelRangeRequest",
     "messageType": "9",
     "protocol": "1",
     "senderRole": "consumer",
     "protocolRoles": "producer,consumer",
     "fields":
     [
         {
         "name": "channelRanges",
         "type": { "type": "array", "items": "Energistics.Datatypes.ChannelData.ChannelRangeInfo"
}
     }
     ]
}
```

3.3.2.10 Message: ChannelStatusChange

Sent from producer to consumer when the status of a channel (as defined in the status field of the ChannelMetadataRecord) has changed. Only sent if the receiveChangeNotification field in ChannelStreamingInfo record used to start this channel is set to True.

Message Type ID: 10

Correlation Id Usage: n/a

Multi-part: False

Sent by: producer

Attribute	Description	Data Type	Min	Max
channelId	The channelId of the channel whose status has changed.	long	1	1
status	The new status of the channel.	ChannelStatuses	1	1

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.ChannelStreaming",
     "name": "ChannelStatusChange",
     "messageType": "10",
     "protocol": "1",
     "senderRole": "producer",
     "protocolRoles": "producer,consumer",
     "fields":
     [
         { "name": "channelId", "type": "long" },
         { "name": "status", "type": "Energistics.Datatypes.ChannelData.ChannelStatuses" }
     ]
}
```

3.3.3 ChannelDataFrame

Stability: 1 - Experimental

The ChannelStreaming protocol is used to send "historical" data, that is, a data "frame" for complete well logs.

ProtocolID: 2

Defined Roles: producer,consumer

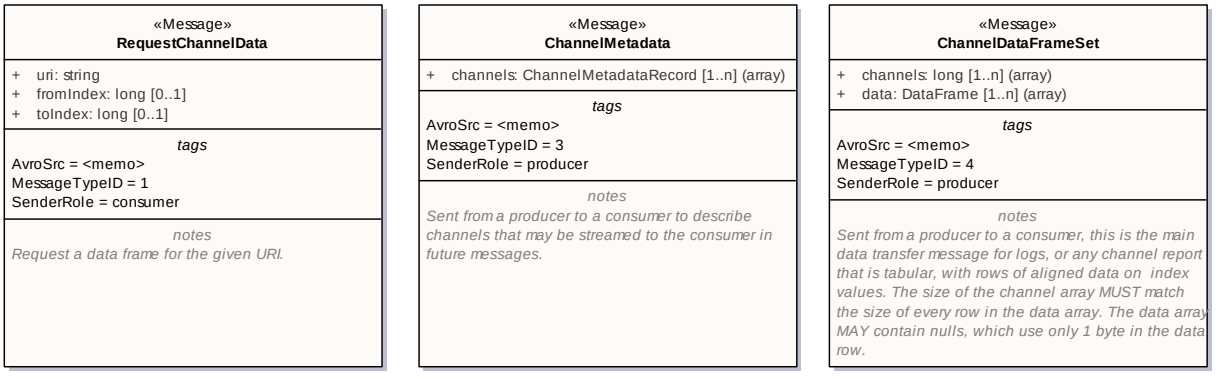

Figure 29: ChannelDataFrame

3.3.3.1 Message: RequestChannelData

Request a data frame for the given URI.

Message Type ID: 1

Correlation Id Usage: n/a

Multi-part: n/a

Sent by: consumer

Attribute	Description	Data Type	Min	Max
uri	The URI of the channel(s) being requested.	string	1	1
fromIndex	Starting value of the primary index.	long	0	1
toIndex	Ending value of the primary index.	long	0	1

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.ChannelDataFrame",
     "name": "RequestChannelData",
     "messageType": "1",
     "protocol": "2",
     "senderRole": "consumer",
     "protocolRoles": "producer,consumer",
     "fields":
     [
         { "name": "uri", "type": "string" },
         { "name": "fromIndex", "type": ["null", "long"] },
         { "name": "toIndex", "type": ["null", "long"] }
     ]
}
```

3.3.3.2 Message: ChannelMetadata

Sent from a producer to a consumer to describe channels that may be streamed to the consumer in future messages.

Message Type ID: 3

Correlation Id Usage: n/a

Multi-part: n/a

Sent by: producer

Attribute	Description	Data Type	Min	Max
channels	An array of channels in the data frame.	ChannelMetadataRecord	1	n

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.ChannelDataFrame",
     "name": "ChannelMetadata",
     "messageType": "3",
     "protocol": "2",
     "senderRole": "producer",
     "protocolRoles": "producer,consumer",
     "fields":
     [
         {
         "name": "channels",
         "type": { "type": "array", "items":
"Energistics.Datatypes.ChannelData.ChannelMetadataRecord" }
     }
     ]
}
```

3.3.3.3 Message: ChannelDataFrameSet

Sent from a producer to a consumer, this is the main data transfer message for logs, or any channel report that is tabular, 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

Sent by: producer

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.	long	1	n
data	Data for the channels specified in the channels element.	DataFrame	1	n

Avro Source

```
{
     "type": "record",
```

```
    "namespace": "Energistics.Protocol.ChannelDataFrame",
    "name": "ChannelDataFrameSet",
    "messageType": "4",
    "protocol": "2",
    "senderRole": "producer",
    "protocolRoles": "producer,consumer",
    "fields":
    [
        {
        "name": "channels",
        "type": { "type": "array", "items": "long" }
    },
        {
        "name": "data",
        "type": { "type": "array", "items": "Energistics.Datatypes.ChannelData.DataFrame" }
    }
    ]
}
```

3.3.4 Discovery

Stability: 3 - Stable

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

ProtocolID: 3

Defined Roles: store,customer

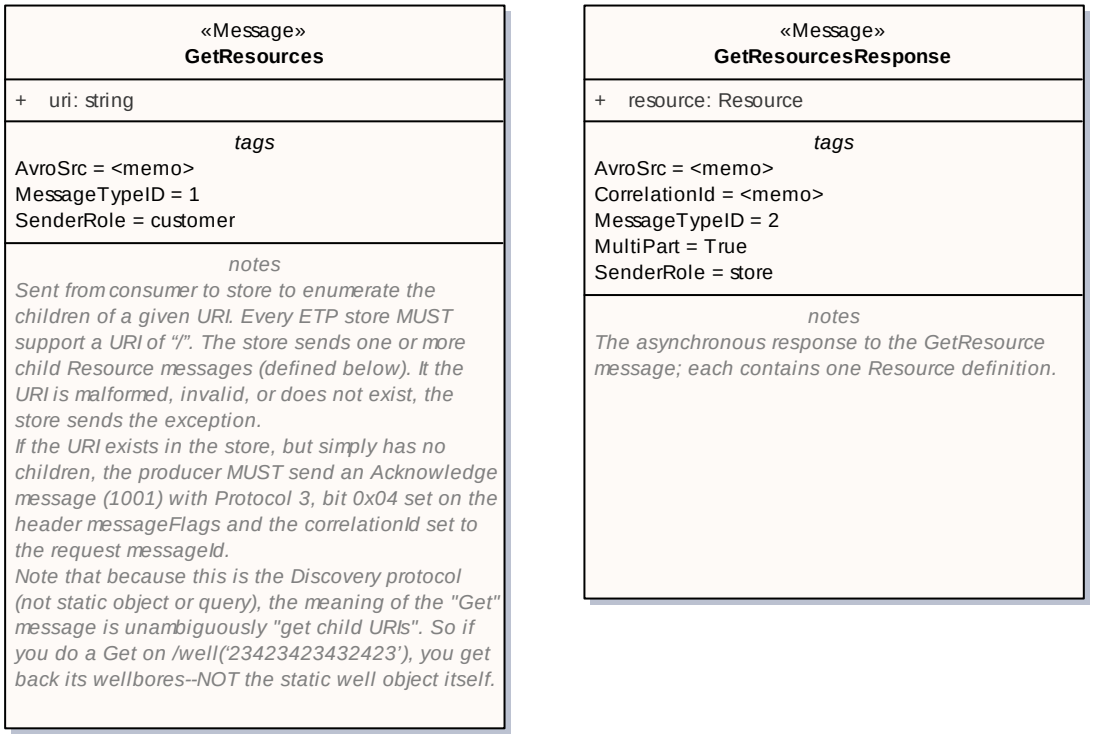

Figure 30: Discovery

3.3.4.1 Message: GetResourcesResponse

The asynchronous response to the GetResource message; each contains one Resource definition.

Message Type ID: 2

Correlation Id Usage:

The messageId of the GetResources message that resulted in this response.

Multi-part: True

Sent by: store

Attribute	Description	Data Type	Min	Max
resource	The resource definition.	Resource	1	1

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.Discovery",
     "name": "GetResourcesResponse",
     "messageType": "2",
     "protocol": "3",
     "senderRole": "store",
     "protocolRoles": "store,customer",
     "fields":
     [
         { "name": "resource", "type": "Energistics.Datatypes.Object.Resource" }
     ]
}
```

3.3.4.2 Message: GetResources

Sent from consumer to store to enumerate the children of a given URI. Every ETP store MUST support a URI of "/". The store sends one or more child Resource messages (defined below). It the URI is malformed, invalid, or does not exist, the store sends the exception.

If the URI exists in the store, but simply has no children, the producer MUST send an Acknowledge message (1001) with Protocol 3, bit 0x04 set on the header messageFlags and the correlationId set to the request messageId.

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 you do a Get on /well('23423423432423'), you get back its wellbores--NOT the static well object itself.

Message Type ID: 1

Correlation Id Usage: n/a

Multi-part: n/a

Sent by: customer

Attribute	Description	Data Type	Min	Max
uri		string	1	1

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.Discovery",
     "name": "GetResources",
     "messageType": "1",
     "protocol": "3",
     "senderRole": "customer",
     "protocolRoles": "store,customer",
     "fields":
     [
         { "name": "uri", "type": "string" }
     ]
}
```

3.3.5 Store

Stability: 2 - Unstable

The Store protocol is used to add, update or delete objects in a store. It is intended to provide similar functionality as the WITSML v1.4.1 Store API.

ProtocolID: 4

Defined Roles: store,customer

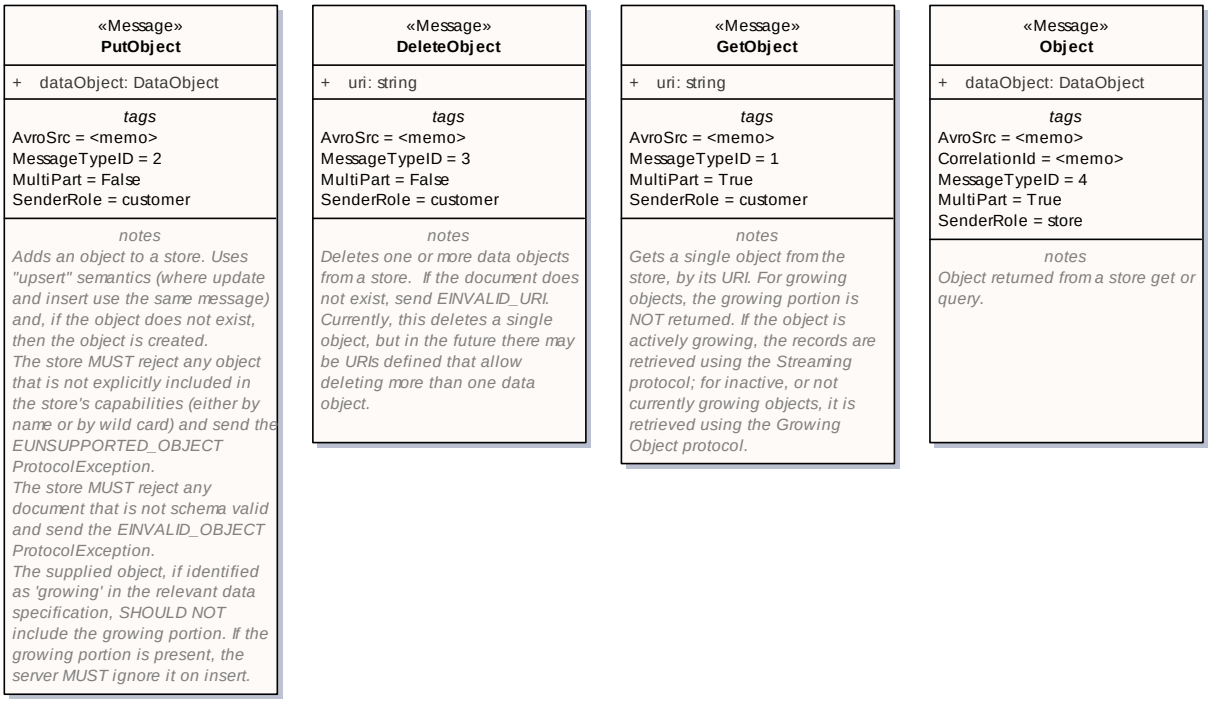

Figure 31: Store

3.3.5.1 Message: GetObject

Gets 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 growing objects, it is retrieved using the Growing Object protocol.

Message Type ID: 1

Correlation Id Usage: n/a

Multi-part: True

Sent by: customer

Attribute	Description	Data Type	Min	Max
uri	The URI of the object to be retrieved.	string	1	1

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.Store",
     "name": "GetObject",
     "messageType": "1",
     "protocol": "4",
     "senderRole": "customer",
     "protocolRoles": "store,customer",
     "fields":
     [
         { "name": "uri", "type": "string" }
     ]
}
```

3.3.5.2 Message: PutObject

Adds an object to a store. Uses "upsert" semantics (where update and insert use the same message) and, if the object does not exist, then the object is created.

The store MUST reject any object that is not explicitly included in the store's capabilities (either by name or by wild card) and send the EUNSUPPORTED_OBJECT ProtocolException.

The store MUST reject any document that is not schema valid and send the EINVALID_OBJECT ProtocolException.

The supplied object, if identified as 'growing' in the relevant data specification, SHOULD NOT include the growing portion. If the growing portion is present, the server MUST ignore it on insert.

Message Type ID: 2

Correlation Id Usage: n/a

Multi-part: False

Sent by: customer

Attribute	Description	Data Type	Min	Max
dataObject	The data object to be updated or added in the store.	DataObject	1	1

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.Store",
     "name": "PutObject",
     "messageType": "2",
     "protocol": "4",
     "senderRole": "customer",
     "protocolRoles": "store,customer",
     "fields":
     [
         { "name": "dataObject", "type": "Energistics.Datatypes.Object.DataObject" }
     ]
}
```

3.3.5.3 Message: DeleteObject

Deletes one or more data objects from a store. If the document does not exist, send EINVALID_URI. Currently, this deletes a single object, but in the future there may be URIs defined that allow deleting more than one data object.

Message Type ID: 3

Correlation Id Usage: n/a

Multi-part: False

Sent by: customer

Attribute	Description	Data Type	Min	Max
uri	The URI of the object to be deleted.	string	1	1

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.Store",
     "name": "DeleteObject",
     "messageType": "3",
     "protocol": "4",
     "senderRole": "customer",
     "protocolRoles": "store,customer",
     "fields":
     [
         { "name": "uri", "type": "string" }
```

```
    ]
}
```

3.3.5.4 Message: Object

Object returned from a store get or query.

Message Type ID: 4

Correlation Id Usage:

The ID of the GetObject message that resulted in this object being returned.

Multi-part: True

Sent by: store

Attribute	Description	Data Type	Min	Max
dataObject	A data object returned from a GetObject message.	DataObject	1	1

Avro Source

```
{
    "type": "record",
    "namespace": "Energistics.Protocol.Store",
    "name": "Object",
    "messageType": "4",
    "protocol": "4",
    "senderRole": "store",
    "protocolRoles": "store,customer",
    "fields":
    [
        { "name": "dataObject", "type": "Energistics.Datatypes.Object.DataObject" }
    ]
}
```

3.3.6 StoreNotification

Stability: 2 - Unstable

The StoreNotification protocol allows Store customers to receive notification of changes to data objects in the Store in an event-driven manner.

ProtocolID: 5

Defined Roles: store,customer

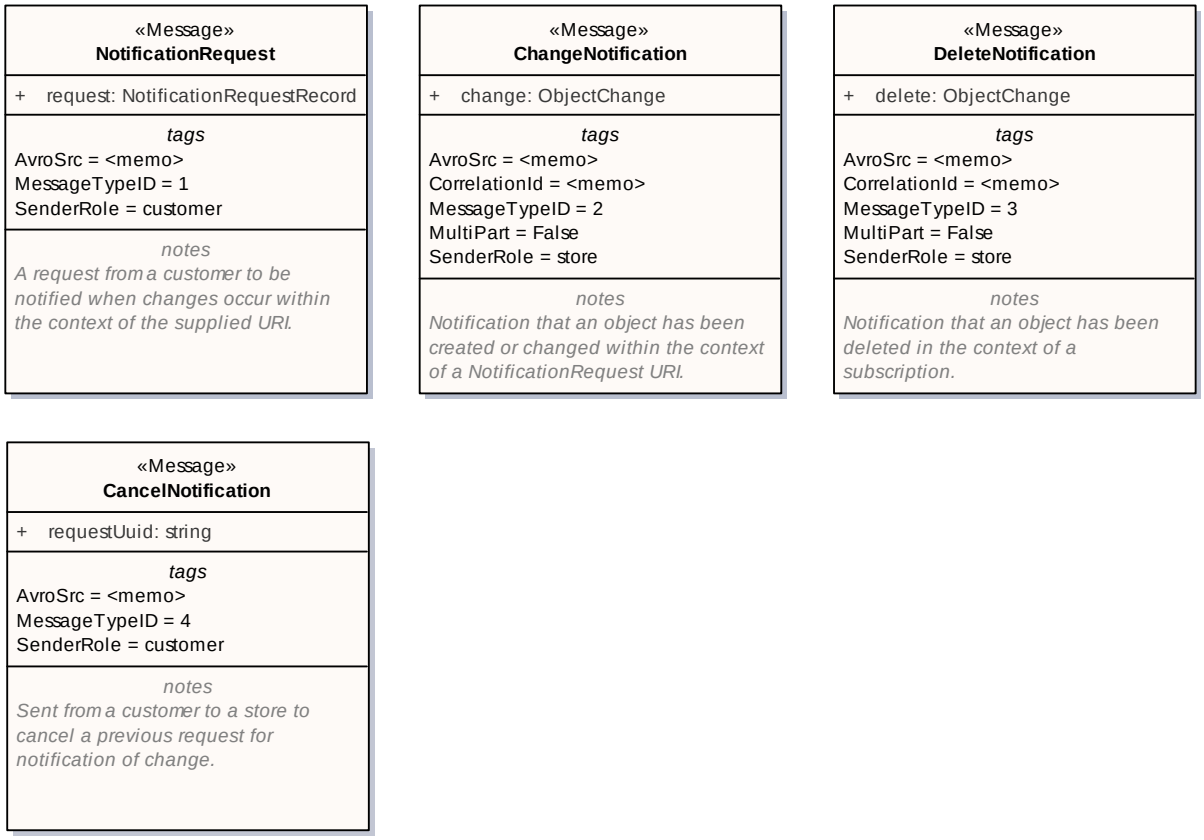

Figure 32: StoreNotification

3.3.6.1 Message: NotificationRequest

A request from a customer to be notified when changes occur within the context of the supplied URI.

Message Type ID: 1

Correlation Id Usage: n/a

Multi-part: n/a

Sent by: customer

Attribute	Description	Data Type	Min	Max
request	The request data.	NotificationRequestRecord	1	1

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.StoreNotification",
     "name": "NotificationRequest",
     "messageType": "1",
     "protocol": "5",
     "senderRole": "customer",
     "protocolRoles": "store,customer",
     "fields":
     [
         { "name": "request", "type": "Energistics.Datatypes.Object.NotificationRequestRecord" }
     ]
}
```

3.3.6.2 Message: ChangeNotification

Notification that an object has been created or changed within the context of a NotificationRequest URI.

Message Type ID: 2

Correlation Id Usage:

The messageId of the original NotificationRequest that caused this notification to occur.

Multi-part: False

Sent by: store

Attribute	Description	Data Type	Min	Max
change	The information describing the change to the data object.	ObjectChange	1	1

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.StoreNotification",
     "name": "ChangeNotification",
     "messageType": "2",
     "protocol": "5",
```

```
    "senderRole": "store",
    "protocolRoles": "store,customer",
    "fields":
    [
        { "name": "change", "type": "Energistics.Datatypes.Object.ObjectChange" }
    ]
}
```

3.3.6.3 Message: DeleteNotification

Notification that an object has been deleted in the context of a subscription.

Message Type ID: 3

Correlation Id Usage:

The messageId of the original NotificationRequest that caused this notification to occur.

Multi-part: False

Sent by: store

Attribute	Description	Data Type	Min	Max
delete	The description of the deleted object. For deleted objects, the object data is not present irrespective of the value of the includeObjectData field in the original request.	ObjectChange	1	1

Avro Source

```
{
    "type": "record",
    "namespace": "Energistics.Protocol.StoreNotification",
    "name": "DeleteNotification",
    "messageType": "3",
    "protocol": "5",
    "senderRole": "store",
    "protocolRoles": "store,customer",
    "fields":
    [
        { "name": "delete", "type": "Energistics.Datatypes.Object.ObjectChange" }
    ]
}
```

3.3.6.4 Message: CancelNotification

Sent from a customer to a store to cancel a previous request for notification of change.

Message Type ID: 4

Correlation Id Usage: n/a

Multi-part: n/a

Sent by: customer

Attribute	Description	Data Type	Min	Max
requestUuid	The UUID of the original notification request.	string	1	1

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.StoreNotification",
     "name": "CancelNotification",
     "messageType": "4",
     "protocol": "5",
     "senderRole": "customer",
     "protocolRoles": "store,customer",
     "fields":
     [
         { "name": "requestUuid", "type": "string" }
     ]
}
```

3.3.7 GrowingObject

Stability: 2 - Unstable

The Growing Object protocol is used to manage the growing parts of data objects that are index-based (i.e., time and depth) but are not appropriate for the ChannelStreaming protocol.

ProtocolID: 6

Defined Roles: store,customer

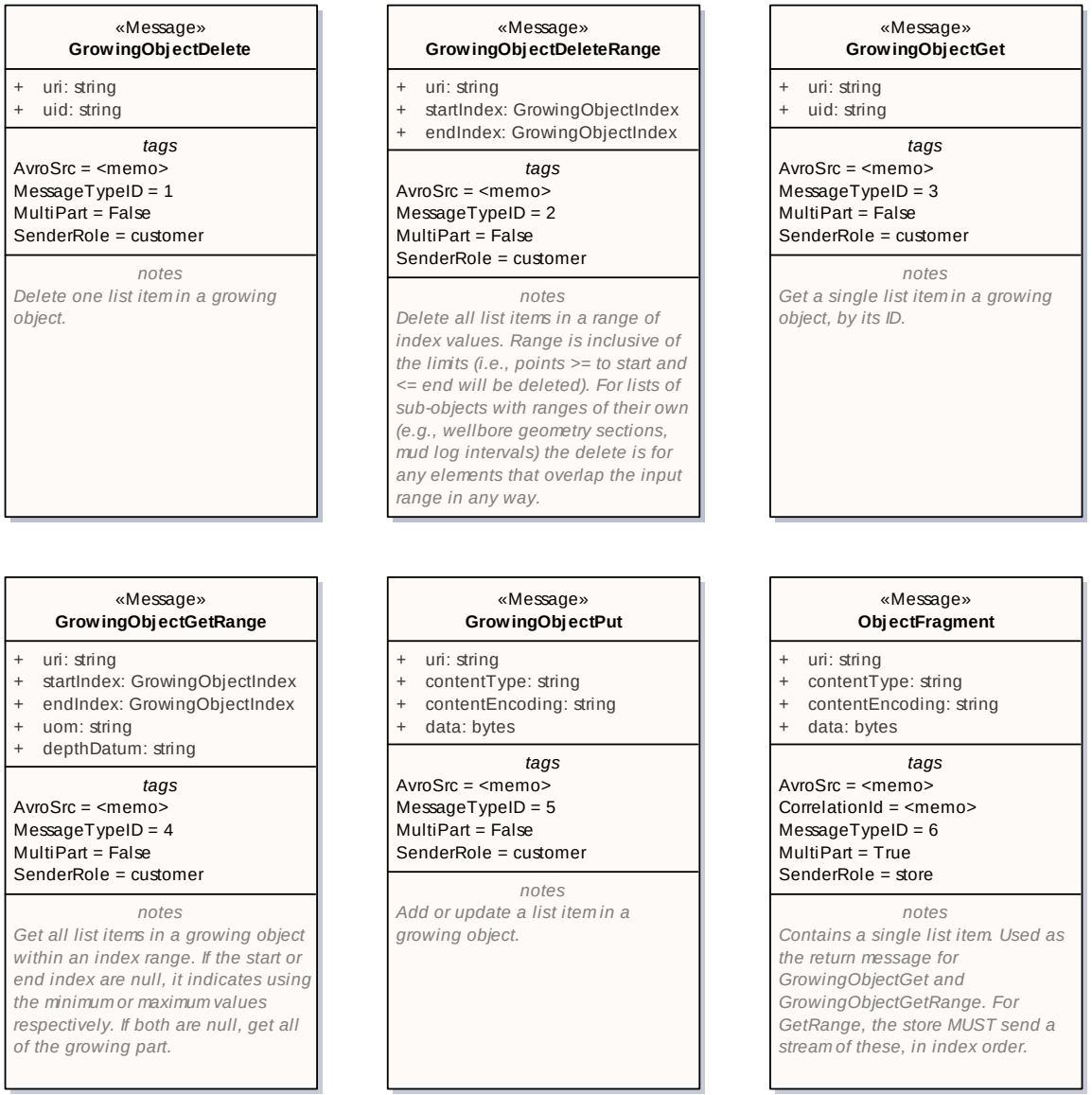

Figure 33: GrowingObject

3.3.7.1 Message: GrowingObjectGetRange

Get all list items in a growing object within an index range. If the start or end index are null, it indicates using the minimum or maximum values respectively. If both are null, get all of the growing part.

Message Type ID: 4

Correlation Id Usage: n/a

Multi-part: False

Sent by: customer

Attribute	Description	Data Type	Min	Max
uri	A URI for the parent object.	string	1	1
startIndex	The starting index for the get range.	GrowingObjectIndex	1	1
endIndex	The ending index for the get range.	GrowingObjectIndex	1	1
uom	A UOM for depth values in the indexes. Defaults to whatever is stored natively in the store.	string	1	1
depthDatum	A depthDatum to be used if the index is an MD. As defined in IndexMetadataRecord.depthDatum.	string	1	1

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.GrowingObject",
     "name": "GrowingObjectGetRange",
     "messageType": "4",
     "protocol": "6",
     "senderRole": "customer",
     "protocolRoles": "store,customer",
     "fields":
     [
         { "name": "uri", "type": "string" },
         { "name": "startIndex", "type": "Energistics.Datatypes.Object.GrowingObjectIndex" },
         { "name": "endIndex", "type": "Energistics.Datatypes.Object.GrowingObjectIndex" },
         { "name": "uom", "type": "string" },
         { "name": "depthDatum", "type": "string" }
     ]
}
```

3.3.7.2 Message: GrowingObjectPut

Add or update a list item in a growing object.

Message Type ID: 5

Correlation Id Usage: n/a

Multi-part: False

Sent by: customer

Attribute	Description	Data Type	Min	Max
uri	A URI for the parent object. The URI MUST resolve to a single data object; if it does not., send EINVALID_URI.	string	1	1
contentType	The content type string for the parent object, as defined in the DataObject record.	string	1	1
contentEncoding	Always text/xml.	string	1	1
data	The data (list items) to be added to the growing object.	bytes	1	1

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.GrowingObject",
     "name": "GrowingObjectPut",
     "messageType": "5",
     "protocol": "6",
     "senderRole": "customer",
     "protocolRoles": "store,customer",
     "fields":
     [
         { "name": "uri", "type": "string" },
         { "name": "contentType", "type": "string" },
         { "name": "contentEncoding", "type": "string" },
         { "name": "data", "type": "bytes" }
     ]
}
```

3.3.7.3 Message: ObjectFragment

Contains a single list item. Used as the return message for GrowingObjectGet and GrowingObjectGetRange. For GetRange, the store MUST send a stream of these, in index order.

Message Type ID: 6

Correlation Id Usage:

The messageId of the GrowingObjectGet or GetRange that resulted in this fragment.

Multi-part: True

Sent by: store

Attribute	Description	Data Type	Min	Max
uri	The URI for the parent object.	string	1	1
contentType	The XML content type of the fragment.	string	1	1
contentEncoding	Currently, this is always "text/xml"	string	1	1
data	The XML data for one part_ wrapper object, returned from a Get or GetRange message, as described in the data field of the DataObject record.	bytes	1	1

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.GrowingObject",
     "name": "ObjectFragment",
     "messageType": "6",
     "protocol": "6",
     "senderRole": "store",
     "protocolRoles": "store,customer",
     "fields":
     [
         { "name": "uri", "type": "string" },
         { "name": "contentType", "type": "string" },
         { "name": "contentEncoding", "type": "string" },
         { "name": "data", "type": "bytes" }
     ]
}
```

3.3.7.4 Message: GrowingObjectDelete

Delete one list item in a growing object.

Message Type ID: 1

Correlation Id Usage: n/a

Multi-part: False

Sent by: customer

Attribute	Description	Data Type	Min	Max
uri	A URI for the parent object. The URI MUST resolve to a single data object; if it does not, send EINVALID_URI.	string	1	1
uid	The ID of the node to be deleted.	string	1	1

Avro Source

```
{
     "type": "record",
```

```
    "namespace": "Energistics.Protocol.GrowingObject",
    "name": "GrowingObjectDelete",
    "messageType": "1",
    "protocol": "6",
    "senderRole": "customer",
    "protocolRoles": "store,customer",
    "fields":
    [
        { "name": "uri", "type": "string" },
        { "name": "uid", "type": "string" }
    ]
}
```

3.3.7.5 Message: GrowingObjectDeleteRange

Delete all list items in a range of index values. Range is inclusive of the limits (i.e., points >= to start and <= end will be deleted). For lists of sub-objects with ranges of their own (e.g., wellbore geometry sections, mud log intervals) the delete is for any elements that overlap the input range in any way.

Message Type ID: 2

Correlation Id Usage: n/a

Multi-part: False

Sent by: customer

Attribute	Description	Data Type	Min	Max
uri	A URI for the parent object. The URI MUST resolve to a single data object; if it does not, send EINVALID_URI.	string	1	1
startIndex	The starting index for the delete range.	GrowingObjectIndex	1	1
endIndex	The ending index for the delete range. MUST be >= startIndex or EINVALID_ARGUMENT.	GrowingObjectIndex	1	1

Avro Source

```
{
    "type": "record",
    "namespace": "Energistics.Protocol.GrowingObject",
    "name": "GrowingObjectDeleteRange",
    "messageType": "2",
    "protocol": "6",
    "senderRole": "customer",
    "protocolRoles": "store,customer",
    "fields":
    [
```

```
        { "name": "uri", "type": "string" },
        { "name": "startIndex", "type": "Energistics.Datatypes.Object.GrowingObjectIndex" },
        { "name": "endIndex", "type": "Energistics.Datatypes.Object.GrowingObjectIndex" }
    ]
}
```

3.3.7.6 Message: GrowingObjectGet

Get a single list item in a growing object, by its ID.

Message Type ID: 3

Correlation Id Usage: n/a

Multi-part: False

Sent by: customer

Attribute	Description	Data Type	Min	Max
uri	A URI for the parent object. The URI MUST resolve to a single data object; if it does not, send EINVALID_URI.	string	1	1
uid	The uid of the element within the growing part. If no such element exists, the store MUST send protocol error ENOT_FOUND.	string	1	1

Avro Source

```
{
    "type": "record",
    "namespace": "Energistics.Protocol.GrowingObject",
    "name": "GrowingObjectGet",
    "messageType": "3",
    "protocol": "6",
    "senderRole": "customer",
    "protocolRoles": "store,customer",
    "fields":
    [
        { "name": "uri", "type": "string" },
        { "name": "uid", "type": "string" }
    ]
}
```

3.3.8 DataArray

Stability: 1 - Experimental

The Data Array protocol is used to transfer large, binary arrays of heterogeneous data values, which Energistics domain standards typically store using HDF5. For example, RESQML uses HDF5 to store seismic, interpretation, and modeling data. As such, the arrays that are transferred with the DataArray protocol are logical versions of the HDF5 data sets.

ProtocolID: 7

Defined Roles: store,customer

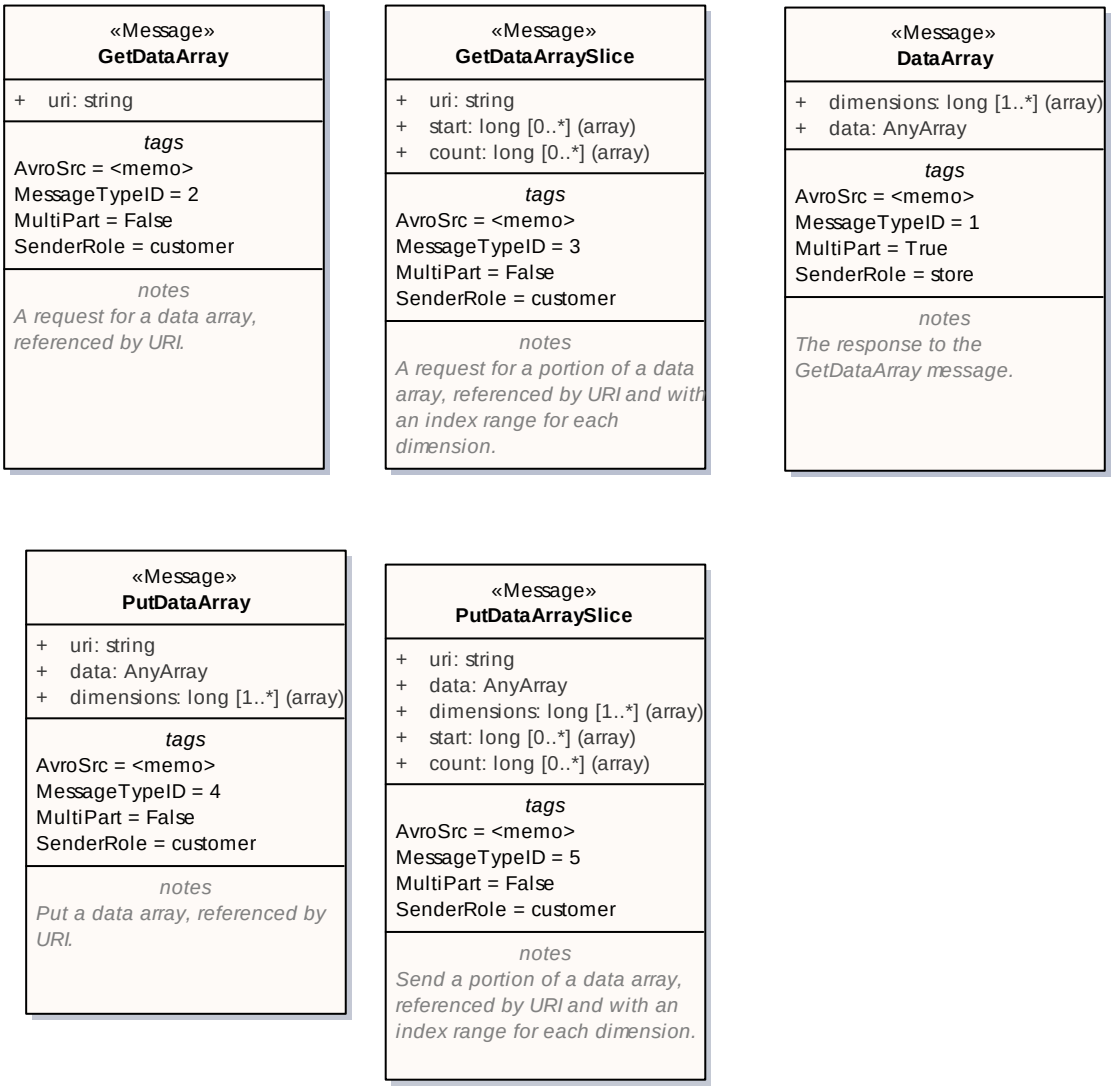

Figure 34: DataArray

3.3.8.1 Message: PutDataArray

Put a data array, referenced by URI.

Message Type ID: 4

Correlation Id Usage: n/a

Multi-part: False

Sent by: customer

Attribute	Description	Data Type	Min	Max
uri	A URI string identifying the array that is sought. This protocol does NOT specify the exact format of the URI; it uses various existing URI conventions. Specifically, for RESQML V2 data sets, the URI is a serialization of the HDF5 proxy as follows: eml://array/resqml20/{uuid of hdf5}/{dataset path}	string	1	1
data	The array data.	AnyArray	1	1
dimensions	An array of dimension sizes for the data array. This MUST be the actual size of the included data, whether or not it is a slice of another array.	long	1	*

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.DataArray",
     "name": "PutDataArray",
     "messageType": "4",
     "protocol": "7",
     "senderRole": "customer",
     "protocolRoles": "store,customer",
     "fields":
     [
         { "name": "uri", "type": "string" },
         { "name": "data", "type": "Energistics.Datatypes.AnyArray" },
         {
         "name": "dimensions",
         "type": { "type": "array", "items": "long" }
     }
     ]
}
```

3.3.8.2 Message: PutDataArraySlice

Send a portion of a data array, referenced by URI and with an index range for each dimension.

Message Type ID: 5

Correlation Id Usage: n/a

Multi-part: False

Sent by: customer

Attribute	Description	Data Type	Min	Max
uri	A URI string identifying the array that is sought. This protocol does NOT specify the exact format of the URI; it uses various existing URI conventions. Specifically, for RESQML V2 data sets, the URI is a serialization of the HDF5 proxy as follows: eml://array/resqml20/{uuid of hdf5}/{dataset path}	string	1	1
data	The array data.	AnyArray	1	1
dimensions	An array of dimension sizes for the data array. This MUST be the actual size of the included data, whether or not it is a slice of another array.	long	1	*
start	The starting index of the sub-array, per dimension.	long	0	*
count	The count of values along each dimension.	long	0	*

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.DataArray",
     "name": "PutDataArraySlice",
     "messageType": "5",
     "protocol": "7",
     "senderRole": "customer",
     "protocolRoles": "store,customer",
     "fields":
     [
         { "name": "uri", "type": "string" },
         { "name": "data", "type": "Energistics.Datatypes.AnyArray" },
         {
         "name": "dimensions",
         "type": { "type": "array", "items": "long" }
     },
         {
         "name": "start",
```

```
        "type": { "type": "array", "items": "long" }
    },
        {
        "name": "count",
        "type": { "type": "array", "items": "long" }
    }
    ]
}
```

3.3.8.3 Message: GetDataArraySlice

A request for a portion of a data array, referenced by URI and with an index range for each dimension.

Message Type ID: 3

Correlation Id Usage: n/a

Multi-part: False

Sent by: customer

Attribute	Description	Data Type	Min	Max
uri	A URI string identifying the array that is sought. This protocol does NOT specify the exact format of the URI; it uses various existing URI conventions. Specifically, for RESQML V2 data sets, the URI is a serialization of the HDF5 proxy as follows: eml://array/resqml20/{uuid of hdf5}/{dataset path}	string	1	1
start	The starting index of the sub-array, per dimension.	long	0	*
count	The count of values along each dimension.	long	0	*

Avro Source

```
{
    "type": "record",
    "namespace": "Energistics.Protocol.DataArray",
    "name": "GetDataArraySlice",
    "messageType": "3",
    "protocol": "7",
    "senderRole": "customer",
    "protocolRoles": "store,customer",
    "fields":
    [
        { "name": "uri", "type": "string" },
```

```
        {
        "name": "start",
        "type": { "type": "array", "items": "long" }
    },
        {
        "name": "count",
        "type": { "type": "array", "items": "long" }
    }
    ]
}
```

3.3.8.4 Message: DataArray

The response to the GetDataArray message.

Message Type ID: 1

Correlation Id Usage: n/a

Multi-part: True

Sent by: store

Attribute	Description	Data Type	Min	Max
dimensions	An array of dimension sizes for the data array. This MUST be the actual size of the included data, whether or not it is a slice of another array.	long	1	*
data	The array data.	AnyArray	1	1

Avro Source

```
{
    "type": "record",
    "namespace": "Energistics.Protocol.DataArray",
    "name": "DataArray",
    "messageType": "1",
    "protocol": "7",
    "senderRole": "store",
    "protocolRoles": "store,customer",
    "fields":
    [
        {
        "name": "dimensions",
        "type": { "type": "array", "items": "long" }
    },
        { "name": "data", "type": "Energistics.Datatypes.AnyArray" }
    ]
```

```
}
```

3.3.8.5 Message: GetDataArray

A request for a data array, referenced by URI.

Message Type ID: 2

Correlation Id Usage: n/a

Multi-part: False

Sent by: customer

Attribute	Description	Data Type	Min	Max
uri	A URI string identifying the array that is sought. This protocol does NOT specify the exact format of the URI; it uses various existing URI conventions. Specifically, for RESQML V2 data sets, the URI is a serialization of the HDF5 proxy as follows: eml://array/resqml20/{uuid of hdf5}/{dataset path}	string	1	1

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.DataArray",
     "name": "GetDataArray",
     "messageType": "2",
     "protocol": "7",
     "senderRole": "customer",
     "protocolRoles": "store,customer",
     "fields":
     [
         { "name": "uri", "type": "string" }
     ]
}
```

3.3.9 WitsmlSoap

Stability: 1 - Experimental

The WitsmlSoap protocol can be used to simplify the use of WITSMLv1.4.1 and earlier over ETP. While it is possible to simply expose both the SOAP and WebSocket endpoints, the use of this protocol makes it easier for clients because they only use a single connection. The messages are an exact transcription of the SOAP messages from the WSDL (which is unchanged since WITSML v1.2).

Note that the expectation is that this protocol will be used in conjunction with Protocol 1 for the realtime services. As such, not all of the WITSML store behaviors need to be supported. In particular, the change log and message objects do not need to be supported.

ProtocolID: 8

Defined Roles: n/a

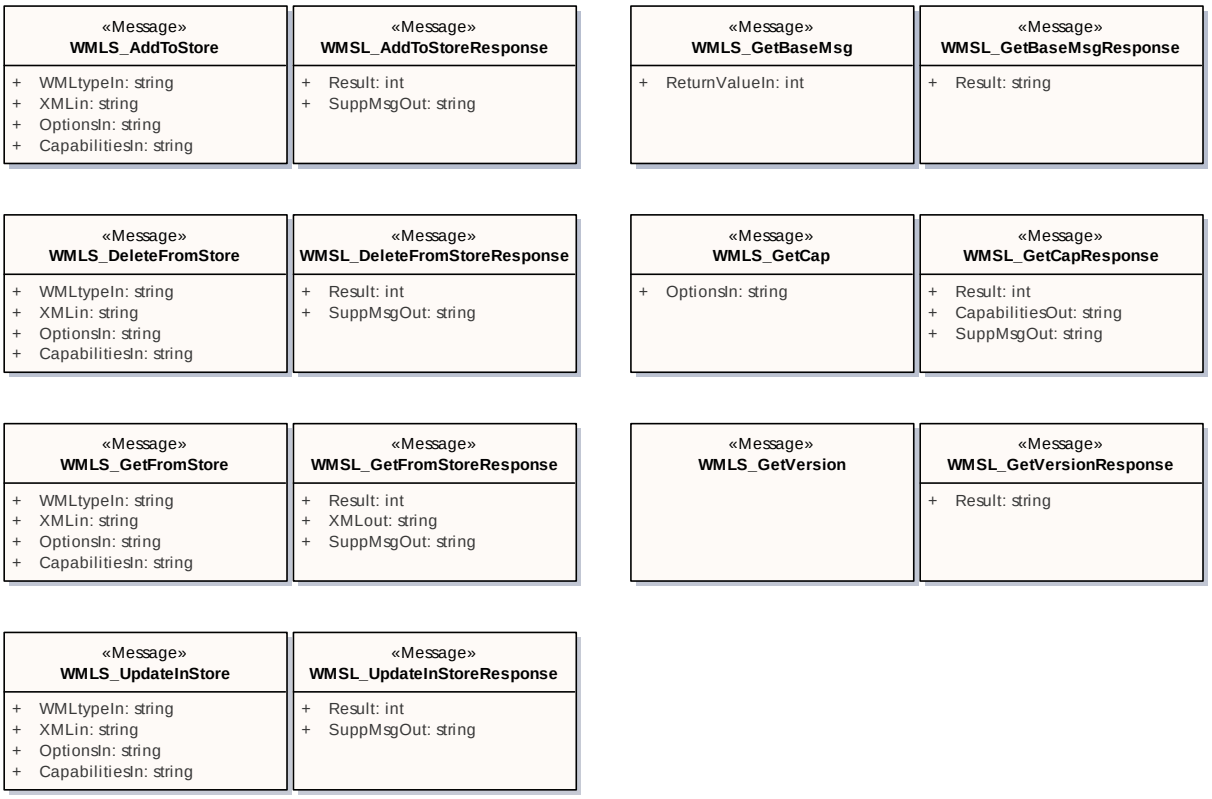

Figure 35: WitsmlSoap

3.3.9.1 Message: WMLS_AddToStore

For definitions of these attributes see the *WITSML Store API*, v1.4.1.1.

Message Type ID: 1

Correlation Id Usage: n/a

Multi-part: False

Sent by: customer

Attribute	Description	Data Type	Min	Max
WMLtypeIn		string	1	1
XMLin		string	1	1
OptionsIn		string	1	1
CapabilitiesIn		string	1	1

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.WitsmlSoap",
     "name": "WMLS_AddToStore",
     "messageType": "1",
     "protocol": "8",
     "senderRole": "customer",
     "protocolRoles": "store,customer",
     "fields":
     [
         { "name": "WMLtypeIn", "type": "string" },
         { "name": "XMLin", "type": "string" },
         { "name": "OptionsIn", "type": "string" },
         { "name": "CapabilitiesIn", "type": "string" }
     ]
}
```

3.3.9.2 Message: WMSL_AddToStoreResponse

Message Type ID: 2

Correlation Id Usage: n/a

Multi-part: False

Sent by: store

Attribute	Description	Data Type	Min	Max
Result		int	1	1
SuppMsgOut		string	1	1

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.WitsmlSoap",
     "name": "WMSL_AddToStoreResponse",
     "messageType": "2",
     "protocol": "8",
     "senderRole": "store",
     "protocolRoles": "store,customer",
     "fields":
     [
         { "name": "Result", "type": "int" },
         { "name": "SuppMsgOut", "type": "string" }
     ]
}
```

3.3.9.3 Message: WMLS_DeleteFromStore

Message Type ID: 3

Correlation Id Usage: n/a

Multi-part: False

Sent by: customer

Attribute	Description	Data Type	Min	Max
WMLtypeIn		string	1	1
XMLin		string	1	1
OptionsIn		string	1	1
CapabilitiesIn		string	1	1

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.WitsmlSoap",
     "name": "WMLS_DeleteFromStore",
     "messageType": "3",
     "protocol": "8",
     "senderRole": "customer",
     "protocolRoles": "store,customer",
     "fields":
     [
         { "name": "WMLtypeIn", "type": "string" },
         { "name": "XMLin", "type": "string" },
```

```
        { "name": "OptionsIn", "type": "string" },
        { "name": "CapabilitiesIn", "type": "string" }
    ]
}
```

3.3.9.4 Message: WMSL_DeleteFromStoreResponse

Message Type ID: 4

Correlation Id Usage: n/a

Multi-part: False

Sent by: store

Attribute	Description	Data Type	Min	Max
Result		int	1	1
SuppMsgOut		string	1	1

Avro Source

```
{
    "type": "record",
    "namespace": "Energistics.Protocol.WitsmlSoap",
    "name": "WMSL_DeleteFromStoreResponse",
    "messageType": "4",
    "protocol": "8",
    "senderRole": "store",
    "protocolRoles": "store,customer",
    "fields":
    [
        { "name": "Result", "type": "int" },
        { "name": "SuppMsgOut", "type": "string" }
    ]
}
```

3.3.9.5 Message: WMLS_GetBaseMsg

Message Type ID: 5

Correlation Id Usage: n/a

Multi-part: False

Sent by: customer

Attribute	Description	Data Type	Min	Max
ReturnValueIn		int	1	1

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.WitsmlSoap",
     "name": "WMLS_GetBaseMsg",
     "messageType": "5",
     "protocol": "8",
     "senderRole": "customer",
     "protocolRoles": "store,customer",
     "fields":
     [
         { "name": "ReturnValueIn", "type": "int" }
     ]
}
```

3.3.9.6 Message: WMSL_GetBaseMsgResponse

Message Type ID: 6

Correlation Id Usage: n/a

Multi-part: False

Sent by: store

Attribute	Description	Data Type	Min	Max
Result		string	1	1

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.WitsmlSoap",
     "name": "WMSL_GetBaseMsgResponse",
     "messageType": "6",
     "protocol": "8",
     "senderRole": "store",
     "protocolRoles": "store,customer",
     "fields":
     [
         { "name": "Result", "type": "string" }
     ]
}
```

3.3.9.7 Message: WMLS_GetCap

Message Type ID: 7

Correlation Id Usage: n/a

Multi-part: False

Sent by: customer

Attribute	Description	Data Type	Min	Max
OptionsIn		string	1	1

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.WitsmlSoap",
     "name": "WMLS_GetCap",
     "messageType": "7",
     "protocol": "8",
     "senderRole": "customer",
     "protocolRoles": "store,customer",
     "fields":
     [
         { "name": "OptionsIn", "type": "string" }
     ]
}
```

3.3.9.8 Message: WMSL_GetCapResponse

Message Type ID: 8

Correlation Id Usage: n/a

Multi-part: False

Sent by: store

Attribute	Description	Data Type	Min	Max
Result		int	1	1
CapabilitiesOut		string	1	1
SuppMsgOut		string	1	1

Avro Source

```
{
     "type": "record",
```

```
    "namespace": "Energistics.Protocol.WitsmlSoap",
    "name": "WMSL_GetCapResponse",
    "messageType": "8",
    "protocol": "8",
    "senderRole": "store",
    "protocolRoles": "store,customer",
    "fields":
    [
        { "name": "Result", "type": "int" },
        { "name": "CapabilitiesOut", "type": "string" },
        { "name": "SuppMsgOut", "type": "string" }
    ]
}
```

3.3.9.9 Message: WMLS_GetFromStore

Message Type ID: 9

Correlation Id Usage: n/a

Multi-part: False

Sent by: customer

Attribute	Description	Data Type	Min	Max
WMLtypeIn		string	1	1
XMLin		string	1	1
OptionsIn		string	1	1
CapabilitiesIn		string	1	1

Avro Source

```
{
    "type": "record",
    "namespace": "Energistics.Protocol.WitsmlSoap",
    "name": "WMLS_GetFromStore",
    "messageType": "9",
    "protocol": "8",
    "senderRole": "customer",
    "protocolRoles": "store,customer",
    "fields":
    [
        { "name": "WMLtypeIn", "type": "string" },
        { "name": "XMLin", "type": "string" },
        { "name": "OptionsIn", "type": "string" },
        { "name": "CapabilitiesIn", "type": "string" }
    ]
```

```
}
```

3.3.9.10 Message: WMSL_GetFromStoreResponse

Message Type ID: 10

Correlation Id Usage: n/a

Multi-part: False

Sent by: store

Attribute	Description	Data Type	Min	Max
Result		int	1	1
XMLout		string	1	1
SuppMsgOut		string	1	1

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.WitsmlSoap",
     "name": "WMSL_GetFromStoreResponse",
     "messageType": "10",
     "protocol": "8",
     "senderRole": "store",
     "protocolRoles": "store,customer",
     "fields":
     [
         { "name": "Result", "type": "int" },
         { "name": "XMLout", "type": "string" },
         { "name": "SuppMsgOut", "type": "string" }
     ]
}
```

3.3.9.11 Message: WMLS_GetVersion

Message Type ID: 11

Correlation Id Usage: n/a

Multi-part: False

Sent by: customer

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.WitsmlSoap",
     "name": "WMLS_GetVersion",
     "messageType": "11",
     "protocol": "8",
     "senderRole": "customer",
     "protocolRoles": "store,customer",
     "fields":
     [

     ]
}
```

3.3.9.12 Message: WMSL_GetVersionResponse

Message Type ID: 12

Correlation Id Usage: n/a

Multi-part: False

Sent by: store

Attribute	Description	Data Type	Min	Max
Result		string	1	1

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.WitsmlSoap",
     "name": "WMSL_GetVersionResponse",
     "messageType": "12",
     "protocol": "8",
     "senderRole": "store",
     "protocolRoles": "store,customer",
     "fields":
     [
         { "name": "Result", "type": "string" }
     ]
}
```

3.3.9.13 Message: WMLS_UpdateInStore

Message Type ID: 13

Correlation Id Usage: n/a

Multi-part: False

Sent by: customer

Attribute	Description	Data Type	Min	Max
WMLtypeIn		string	1	1
XMLin		string	1	1
OptionsIn		string	1	1
CapabilitiesIn		string	1	1

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.WitsmlSoap",
     "name": "WMLS_UpdateInStore",
     "messageType": "13",
     "protocol": "8",
     "senderRole": "customer",
     "protocolRoles": "store,customer",
     "fields":
     [
         { "name": "WMLtypeIn", "type": "string" },
         { "name": "XMLin", "type": "string" },
         { "name": "OptionsIn", "type": "string" },
         { "name": "CapabilitiesIn", "type": "string" }
     ]
}
```

3.3.9.14 Message: WMSL_UpdateInStoreResponse

Message Type ID: 14

Correlation Id Usage: n/a

Multi-part: False

Sent by: store

Attribute	Description	Data Type	Min	Max
Result		int	1	1
SuppMsgOut		string	1	1

Avro Source

```
{
     "type": "record",
     "namespace": "Energistics.Protocol.WitsmlSoap",
     "name": "WMSL_UpdateInStoreResponse",
     "messageType": "14",
     "protocol": "8",
     "senderRole": "store",
     "protocolRoles": "store,customer",
     "fields":
     [
         { "name": "Result", "type": "int" },
         { "name": "SuppMsgOut", "type": "string" }
     ]
}
```

4 Security

In any communication protocol—especially one carrying sensitive, private data—security is a major concern. From RFC6455, we can see that the WebSocket specification itself does not provide much guidance:

> 10.5. WebSocket Client Authentication
>
> This protocol doesn't prescribe any particular way that servers can authenticate clients during the WebSocket handshake. The WebSocket server can use any client authentication mechanism available to a generic HTTP server, such as cookies, HTTP authentication, or TLS authentication.

In general, the approach for ETP has been similar, in that it does not define any new security protocols. Rather, it relies on the available security mechanisms in its underlying protocols (HTTP, WebSocket, TLS, TCP, etc.). As in WITSML, ETP specifies authentication methods which MUST be implemented by all servers for interoperability, but allows for additional methods (such as client certificates, for example) to be used by agreement between specific parties.

This part of the ETP specification is primarily concerned with the exact mechanism for authenticating the WebSocket upgrade. It is does not attempt to define all of the methods that may be used to acquire credentials, federate identity, etc.

ETP provides two well-known authentication mechanisms:

- JSON Web Tokes (JWT) token-based authentication (MUST be supported)
- Basic authentication

All ETP servers MUST implement JWT. Implementation guidelines for specific domains (such as WITSML v2.0) may provide more strict language that requires one or both of these methods to be supported

Specific vendors, service companies, and operators MAY also implement any other appropriate security mechanisms (such as SAML tokens), but they are not required by the specification and may lead to interoperability issues.

In all cases, the client SHOULD use the Authorization request header defined by HTTP/1.1. Servers MUST support this method. For browser-based clients, it is not possible to add request headers to the upgrade request. This is because the HTML5 WebSocket API definition does not allow access to the request headers. For this reason, the client MAY provide the information in the query string of the upgrade request. Servers MUST also support this method. If the information is provided in both the request headers and in the query string, the server MUST use the request header.

When provided on the query string, the authentication information MUST be URL-encoded. The query variable is Authorization and the: character is replaced with the equal sign (=). So the following authorization header:

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ
```

Would appear as follows in the query string:

```
Authorization=Basic%20QWxhZGRpbjpvcGVuIHNlc2FtZQ
```

4.1 JWT Token-based Authentication

All ETP servers MUST support authentication using JWT, as described in RFC 7519 (https://tools.ietf.org/html/rfc7519) (for a non-normative introduction to JWT, refer to https://jwt.io/introduction/). The token is presented during the WebSocket upgrade request, using either the Authorization header or on the query string as described above. ETP mandates the use of JWT, but does not specify how the token is obtained. This will be implementation dependent and may be different for user-oriented and machine-oriented clients.

The following additional restrictions are placed on tokens used with an ETP connection:

4.1.1 Authentication Schema

The token MUST be presented using the Bearer schema. The content of the header should look something like this:

```
Authorization: Bearer <token>
```

4.1.2 Signing Algorithm

All tokens MUST be signed using the HS256 algorithm. The rationale here is that JWT allows for some very insecure signing algorithms (including 'none'). By standardizing on a particular signing algorithm, it will be much easier to ensure that all ETP servers and clients can communicate in a secure and standard way. Thus the header MUST always be:

```
'{"alg":"HS256","typ":"JWT"}'
```

4.1.3 Required Claims

The following claims are REQUIRED for an ETP token:

- Issuer (iss)
- Subject (sub)
- Issued at (iat)
- Expiry time (exp)

4.1.4 Token Expiry and Renewal

- If the iat claim is in the future at the time of the upgrade request, the server MUST reject the request.
- If the exp claim is in the past at the time of the upgrade request (i.e., the token is already expired) the server MUST reject the request.
- If the security token used to establish the session expires, the server MAY disconnect the session at any time. The server MUST NOT use a CloseSession message to do so. Before disconnecting the session, the server MUST send the EXPIRED_TOKEN ProtocolException.

  Disconnection for this reason is considered an abnormal session termination, and the session survivability behaviors (as defined in this specification) MUST be observed. Observing these behaviors allows the client 1 hr. to reconnect the session with a valid token, without losing any session state.

4.1.5 Token Verification

Typically, a software library is used to decode and validate the JWT. While the specifics of this process depend on the library used and the details of the token provider, the following framework identifies the steps the server must take. A failure at any step would cause the request to be denied.

1. The token string MUST parse as valid JWT.
2. The signing algorithm MUST be HS256.
3. The signature MUST be verified using the signing secret or key.
4. The claims are verified. Apart from the iat and exp as discussed above, ETP does not specify how the server verifies other claims, such as audience and subject.

4.2 Basic Authentication

Basic authentication is as described in RFC-2617 (https://www.ietf.org/rfc/rfc2617.txt) for HTTP connections. As in HTTP, basic authentication has many security issues, especially when used on an insecure connection (i.e., not SSL).

4.3 Other considerations

4.3.1 Unauthenticated Requests

An ETP server MUST reject unauthenticated upgrade requests with HTTP 401.

4.3.2 Session State

It is specifically recommended that cookies, or any other form of HTTP session state, NOT be trusted to authenticate the upgrade request.

4.3.3 Origin Header

The Origin header, specifically, should NOT be trusted. While it is relatively secure when sent from a browser agent, WebSocket upgrade requests can (and in the case of ETP most often *will*) come from server or desktop agents. These agents use APIs that allow any value at all to be set on the HTTP headers. Thus, there is no guarantee that the Origin header is actually correct. That said, if the Origin header is NOT something that is expected, then you have good reason to mistrust and terminate the connection.

4.4 Use of Secure WebSocket

It is highly recommended that all production ETP servers use the secure WebSocket protocol (i.e., wss), which allows for end-to-end encryption of all ETP traffic. Companion specifications for specific XML families and contracts between vendors and customers may require this.

5 Server Capabilities

As described in section 2.1.1, the session-initiation process provides for a negotiation of the protocols and object types that will be exchanged in a session. ETP also provide a mechanism for clients to "pre-discover" the capabilities of a server out-of-band of the WebSocket connection, using a simple HTTP GET of a JSON object. This mechanism is not strictly required for every ETP server, but certain implementations of ETP for specific workloads and XML standards may require it to be implemented (for example, it may be required for a WITSML store). Use of these mechanisms will be documented in the companion implementation specifications for each Energistics ML.

- The URL for the capabilities document can be found by appending the string 'well-known/etp-server-capabilities' to the HTTP-equivalent URL of the ETP WebSocket end point. For example, if you have an ETP server listening at wss://etp.sample.org:8080, then the server capabilities document could be retrieved with an HTTP GET from https://etp.sample.org:8080/.well-known/etp-server-capabilities.
- The capabilities document MUST match 'TLS-ness' of the WebSocket connection. That is, if the WebSocket address is at ws://, then the document MUST be at http://. If the WebSocket is at wss://, then the document MUST be at https://.
- The returned resource MUST have a content type of **application/json**.
- The contents of the returned JSON object MUST match an Avro JSON serialization of the Energistics.Datatypes.ServerCapabilities structure defined in section 3.2.2.
- Additional fields may be present in the JSON document, but they are not part of the standard.
- The capabilities document MAY be a protected resource, but it is not required to be so.

Appendix A. Error Codes

The following is a summary table of all ETP error codes. Note that the Mnemonic is informational only, and is used in the documentation to make references to error codes more readable. but it has no meaning on the wire (for either binary or json encodings). Implementors may wish use these terms as a #define or constant name in their code, but this is not a part of the spec either. Additionally, it is suggested the errorMessage field in the ProtocolException message contain the Mnemonic as a part of the text description of the error.

Scope	ErrorCode	Mnemonic	Notes
All	1	ENOROLE	The agent does not support the requested role.
All	2	ENOSUPPORTEDPROTOCOLS	The server does not support any of the requested protocols.
All	3	EINVALID_MESSAGETYPE	The message type ID is invalid for the given protocol.
All	4	EUNSUPPORTED_PROTOCOL	The agent does not support the protocol identified in a message header.
All	5	EINVALID_ARGUMENT	Logically invalid argument.
All	6	EPERMISSION_DENIED	Permission denied.
All	7	ENOTSUPPORTED	The operation is not supported by the agent.
All	8	EINVALID_STATE	Indicates that the message is not allowed in the current state of the protocol. For example, sending ChannelStreamingStart for a channel that is already streaming.
All	9	EINVALID_URI	Sent from a producer to a consumer when a request is made to describe a URI that does not exist on the producer.
All	10	EEXPIRED_TOKEN	Sent from server to client when it is about to terminate the session because of an expired security token.
All	11	ENOT_FOUND	Used when a resource is not found.
ChannelData	1002	EINVALID_CHANNELID	Sent from a producer to a consumer when operations are requested on a channel that does not exist.
Store	3001	EUNSUPPORTED_OBJECT	Sent in the Store protocols, when either role sends or requests a data object type that is not supported, according to the protocol capabilities.
Store	3002	EINVALID_OBJECT	Sent in the Store protocols, when either role sends an invalid XML document. Note: We do not distinguish between well-formed and invalid for this purpose. The same error message is used in both cases.
Store	3003	ENOCASCADE_DELETE	This exception is sent when an attempt is made to delete an object that has children and the server does not support cascading deletes.
Store	3004	EPLURAL_OBJECT	Sent when an agent uses puts for more than one object under the plural root of a 1.x Energistics data object. ETP only supports a single data object, one XML document.
Store	3005	EGROWING_PORTION_IGNORED	Sent from a store to acustomer when the customer supplies the growing portion in a Put. This is advisory only; the object is upserted, but the growing portion is ignored.