WITSML STORE
Application Program Interface (API)

Version 1.4.1

WITSML	Data-object definitions and a Web services specification for the right-time, seamless flow of well data between operators and service companies to speed and enhance decision making.
Version:	**WITSML STORE Application Program Interface (API) Version 1.4.1**
Abstract:	This document describes the components of the WITSML API. The components consist of the client/server Web service interfaces.
Prepared by:	Energistics and the WITSML SIG
Date published:	1 August 2012
Document type:	formal specifications
Keywords:	standards, energy, drilling, web service

Document Information	
DOCUMENT VERSION:	DOC1.4.1-v2
DATE:	1 August 2012
Technical	Color: R:210, G:124, B:50
Language	US English

Amendment History			
API Version	**Date**	**Comment**	**By**
1.4.1	1 August 2012	Minor modifications and bug fixes of 1.4.1 release. (Data schema versions for this release are 1.4.1.1.) • Updated enumvalues file for objectGroup data-object to include value of “other” and use of groupSubType element (Section 2.1.1.2). (WITSML Change Request WCR-2012-0025) • Corrected inconsistency. If in GetCap function, the client doesn’t send a data schema version, the server gives an error code (Sections 6.3.2 and 6.3.3) (WCR-2012-0022) • Corrected verbiage around error message for multiple queries in GetFromStore function (Section 6.6.4). (WCR-2012-0027) • Numbered standard queries (e.g., SQ-001, SQ-002) for easy reference (Section 6.6.5.1). (WCR-2012-0007) • Corrected XMLout example for standard query “Get details for a well” (Section 6.6.5.4). (WCR-2012-0024) • Corrected XMLout example for standard query “Get log” (6.6.5.16). (WCR-2012-0023) • Added second example query (Query 2) for objectGroup data-object with type = “other”. (Section 14.2.6). (WCR-2012-0025)Corrected XML examples for spacing and UID formats (Chapter 4 (various sections) and Section 6.6.5, Standard Query Templates.) (WCR-2012-0028) • Corrected text regarding QueryIn parameters and requestObjectSelectionCapability to reference Minimu Query template instead of "empty" query. (See Section 6.6.2, page 54 and Section 6.6.2.1, page 57). Discussed and agreed upon as part of the Candidate Release review. • Standard Query Table (Section 6.6.5.1, page 67) for column heading Return Elements/Attributes: Removed note from SQ-001 and moved to column heading; deleted previous note in column heading. Discussed and agreed upon as part of the Candidate Release review. • Re-numbered pages so that title page of manual begins on page *1* (not on page *i* as in previous version of this API document) so that page numbers are consistent between the published PDF document and the Microsoft Word document. • Clean up of minor errors, spacing issues, etc. in various examples in the document. • Corrected minor typos reported by Certification Team and others.	Energistics/ WITSML SIG
1.4.1	23 Sept 2011	Major changes to the standard and major re-write of the API document.	Energistics/ WITSML SIG
1.4.0	12 Nov 2008	Not a production version.	Energistics
1.3.1	9 Dec 2005		Energistics

1.3.0	16 Mar 2005	Not a production version.	Energistics
1.2	28 Feb 2003		NPSi

Table of Contents

1. Introduction

1.1 WITSML Overview

The Wellsite Information Transfer Standard Markup Language (WITSML) consists of XML data-object definitions and a Web services specification developed to promote the right-time, seamless flow of well data between operators and service companies to speed and enhance decision making.

WITSML is consists of:

- **Data schemas** that define data-objects (such as wells, wellbores, logs, etc.) and their relationship to one another, for other data-objects related to drilling. For a complete list of the main data-objects in WITSML, see APPENDIX C: WITSML Data-objects, page 115.
- **enumValues.xml**. A loader file that contains tables for defining valid values for data elements, such as units of measure and lithology types.
- **STORE API** (this document) describes functions and behaviors of WITSML servers and how client applications interact with those servers.

  All files are available in the WITSML download at http://www.energistics.org/drilling-completions-interventions/witsml-standards/current-standards. For more information on WITSML data-objects and schemas, see Section 2.1, page 13.

The STORE API described in this document specifies a standardized way of electronically transporting WITSML data-objects between systems (clients and servers) using HTTP/S-based protocols. However, it is acceptable for companies to agree to send and receive WITSML data streams without the use of the WITSML API.

The WITSML API is platform independent.

For a list of reference standards, see APPENDIX E: References, page 155.

1.1.1 Example Implementation

Implementation of the WITSML specification can include the components listed below and shown in Figure 1. The data schemas and enumeration table file are the minimum required. Other components are optional.

- **WITSML data schemas** and **enumerated values file (required).**
- **WITSML server** is software designed according to this API specification to process retrievals, additions, deletions, or updates of WITSML data as defined by WITSML schemas.
- **Client applications** are software programs designed to understand/use the WITSML data schemas and to interact with WITMSL servers. This API document also distinguishes between client application and server responsibilities.
- **Simple data-object Access Protocol (SOAP)** is used to transport the requests/responses between the client and server.

For examples of how WITSML may be implemented, see APPENDIX D: Examples, Section 14.1, page 118.

1.2 Specification Organization

- All schema files (downloaded from the Energistics website) contain the normative definition of the XML data-objects and structures. If this API document and a schema differ on the name of an element or attribute in the schema, then the schema defines the normative name.
- This API document contains the normative definition of any behavior associated with the information in the schemas.

Example WITSML Implementation

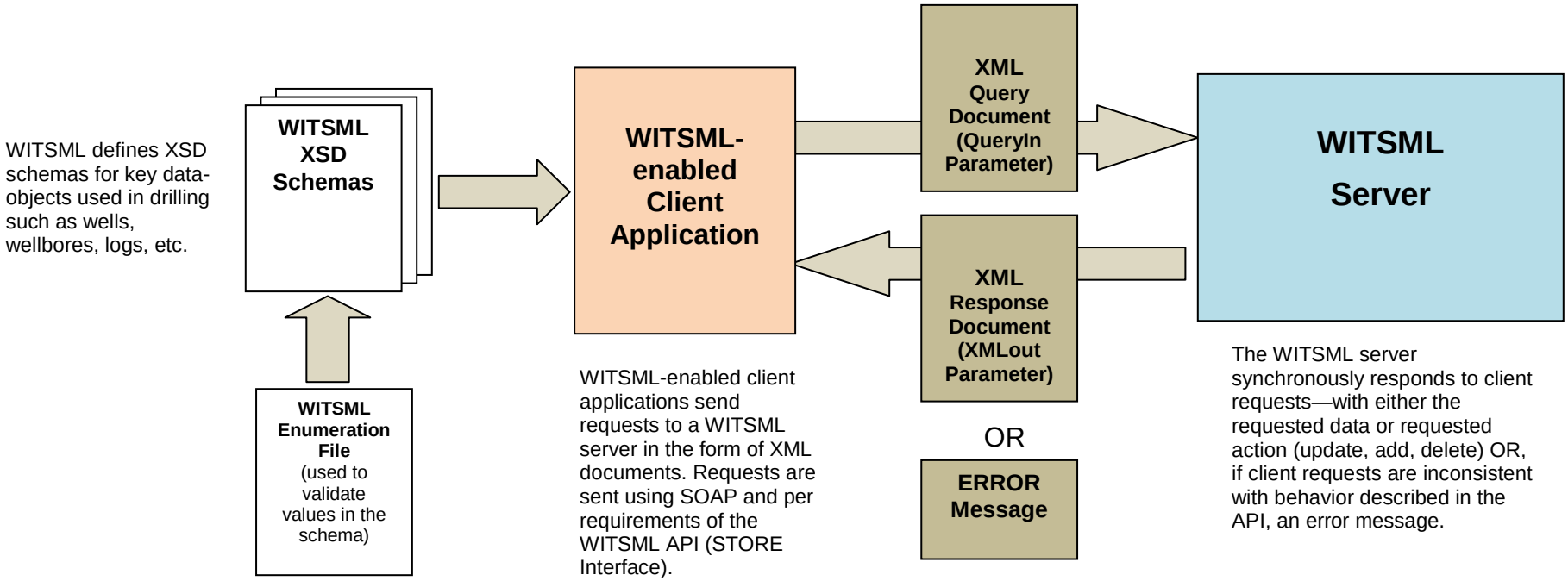


Figure 1 Overview of key WITSML components and interaction of WITSML-enabled clients and a WITSML server.

1.3 Audience, Purpose, and Scope

This specification:

- Is intended for IT professionals such as software engineers, programmers, and others who are developing WITSML servers and/or client applications that can read, writem, delete, and update data on a WITSML server.
- Assumes the audience has knowledge of XML and developing client-server applications using XML and HTTP/S.
- Defines the functions and behaviors of a WITSML server and the client application software that interacts with a WITSML server.
- Describes only the external interfaces exposed by the API, not any particular implementation. The internal implementation of the API can and will vary. All references in this document to implementation scenarios are for example only.

To ensure you are reading the latest version of this document, visit the Energistics website, http://www.energistics.org/drilling-completions-interventions/witsml-standards/current-standards.

1.4 Documentation Conventions

This section lists key conventions used in this documentation that will help you understand expected behaviors and responsibilities.

1.4.1 Specification of Required Server Behavior

Required server behavior is indicated throughout this document as:

The server MUST…

All behaviors specified with this convention are used to develop server certification testing.

1.4.2 Specification of Required Client Behavior

Required client behavior is indicated throughout this document as:

The client MUST ***[else error -9999]***…or *A query or query template MUST* ***[else error -9999]***

In this example, *-9999* is a placeholder for actual error codes (return values) that a server MUST return, IF a server implements validation and detects the condition. Note that a server may not detect certain client compliance failures and may not issue an error when a client violates a specific MUST condition.

If a server does return an error code, it MUST be one of the codes listed in this document, as specified by the API behavior.

For a complete list of return values (error codes) and their associated messages, see APPENDIX A: Defined Return Values, page 110.

1.4.3 Other Document Conventions

Item	Convention
Links	Though no special text formatting is used, all chapter, section and page numbers in this document are hyperlinks
Boolean flags	For Boolean flags, XML recognizes both “true” and “1” and “false” and “0”. To be consistent with XML, WITSML also recognizes both values. However, for simplicity in this document, only “true” and “false” are used.

1.5 Overview of Changes since the Previous API Version

This section provides an overview of key changes since the previous production release of WITSML, which was version 1.3.1 in 2005.

High-level design goals for WITSML 1.4.1 included simplifying the design and reducing ambiguity in this API document, which had resulted in inconsist behavior across WITSML servers. In support of these goals, changes have been made to schemas and this API document, which include:

- Removed the wellLog data-object. Functionality has been combined with the log data-object, for a cleaner, simpler design. To replace the multiple log capabilities of the wellLog data-object, a new data-object, named *objectGroup,* provides a way to group logs together, for example, for multi-pass well testing (see Section 2.1.1.2, page 14).
- Removed the realTime data-object, replacing it with the simplified functionality of a requestLatestValues option (see Section 6.6.2.1, page 55).
- Added a changeLog object, a server object designed to help clients more easily determine which objects in a WITSML server have changed. Clients can query a single object type, the changeLog object, (instead of each individual data-object), which also helps reduce server load (see Section 9.3.4, page 103).
- Specified clear data truncation results behavior so that servers can better manage the volume of data returned and clients have a clear way to request the additional data (see the sub-section titled "If a Server Limits Returned Data," page 63).
- Identified and developed templates for standard queries, which must be supported by all WITSML v1.4.1 servers (see Section 6.6.5, page 65).
- Added support for data compression (see Section 3.3, page 18 and Section 4.2.4.1, page 30).
- Added a required mechanism for a client to identify itself (software name) to a server using the HTTP user-agent field (see Section 4.2.2, page 28).
- Developed and refined standard error messages (or return values), which provide more explicit descriptions of errors within the context of specific WITSML functions (See APPENDIX A: Defined Return Values, page 110).
- Revised this API document to clearly state server behavior, client behavior (in interacting with a WITSML server) and related error messages. Implemented standard language conventions for specifying this behavior (see Section 1.4, page 10). Added more mandatory server behavior to help improve operational consistency across WITSML servers.
- Revised this API document, including organization, format/layout, and language. Added more cross references within the sections with links to related content.

1.6 History of WITSML

WITSML was initially developed in October 2000 by the WITSML project, an oil industry initiative sponsored by BP and Statoil, as a new standard for drilling information transfer to evolve WITS, an earlier standard. Initial participants included Baker Hughes, Landmark (Halliburton), Schlumberger, and NPSi (as technical advisor).

At the completion of WITSML v1.2 in March 2003, Energistics (known then as POSC) accepted custody of WITSML and is managing the support and future evolution of WITSML through the WITSML Special Interest Group (SIG).

WITSML was designed to be a more modern alternative to WITS (Wellsite Information Transfer Standard). Some of the original semantic content of the WITSML data schemas was derived from the WITS specification.

1.7 Future Plans

The user community for the WITSML standard is the Energistics WITSML SIG. The SIG has ongoing activities to monitor WITSML use and to identify, refine, and agree on recommendations for

enhancements. Energistics and the SIG work together on a formal version management process that can lead to future versions of the WITSML standard being planned, developed, and released.

2. Key Concepts

2.1 WITSML Data-objects

A WITSML data-object is a logical representation and organization of the data items associated with the major components and operations involved in drilling a well. For example, a data-object known as the rig data-object contains data items related to a rig, such as the rig's owner, type, and manufacturer. For a complete list of WITSML data-objects, see APPENDIX C: WITSML Data-objects, page 115.

Key characteristics of WITSML data-objects are listed here. The sections that follow provide more details.

- A WITSML `data-object` is not a formal programming object (with methods, properties, events, etc.).
- The data-objects and the relationship between them (e.g., parent-child relationship) are defined by the WITSML data schemas and associated static enumeration tables, such as units of measure, lithology, and others.
- When being exchanged between systems or software applications, each of the logical data-objects is represented as a physical XML document as defined by the WITSML data schemas.
- Each WITSML data-object has an identifier, and the identifier(s) of its ancestry or the hierarchy by which it is derived. For instance, a mud log (mudLog) data-object has an identifier of itself, and the identifiers of its parent (wellbore) data-object and its grandparent (well) data-object (Figure 2).

2.1.1 Data-object Organization

Data-objects are organized into a single hierarchical "tree," with the well data-object at the top of the tree. The well data-object is the parent of one or more wellbore data-objects (child data-object); a wellbore data-object can have one or more children, such as mud logs, trajectories, and so on. Data-objects are also assigned identifiers (UID in Figure 2), which are explained in the next section.

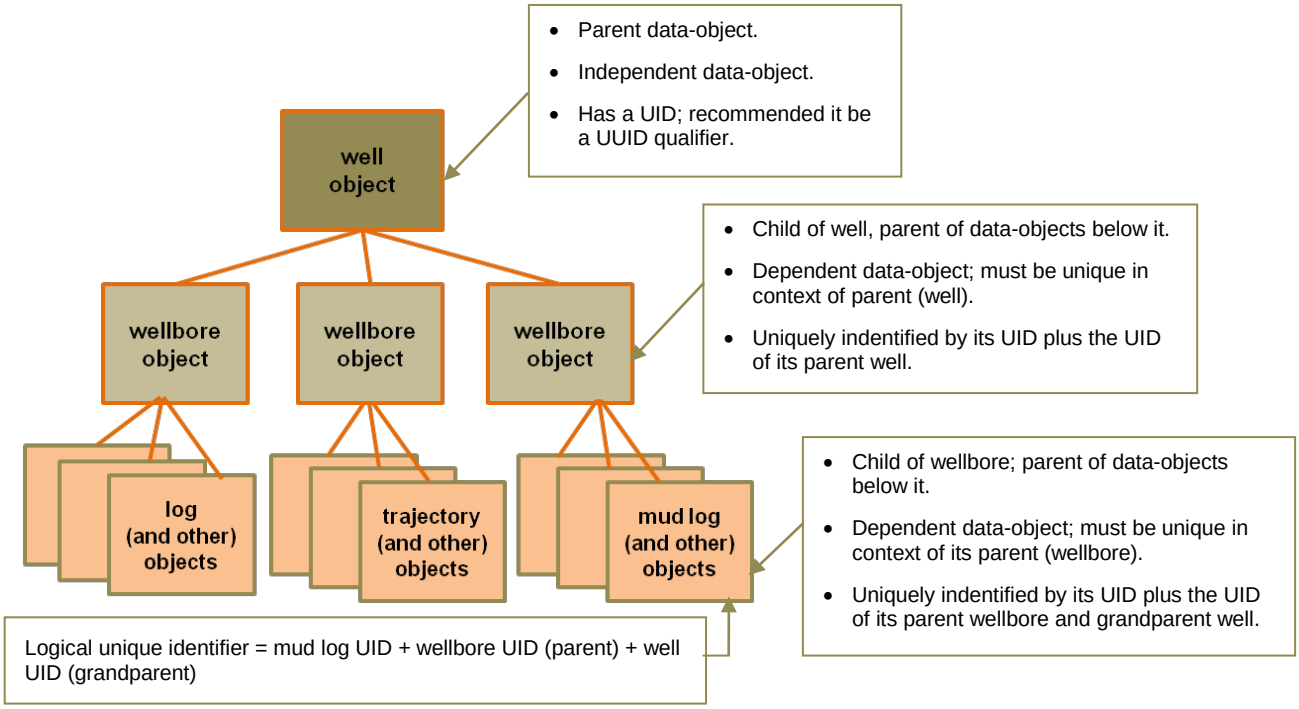


Figure 2 WITSML data-object relationships: Data for wells, wellbores and other related data-objects, such as mud logs, are normalized to create a parent-child hierarchical relationship. (The following pages provide more information about concepts in this figure.)

2.1.1.1 Growing Data-objects

To meet the needs of drilling operations, WITSML defines growing data-objects, which, in some cases, must be handled differently than standard WITSML data-objects.

WITSML defines two types of growing data-objects:

Type of Growing data-object	Definition
Random	The occurrences of data are unrelated and might overlap or coexist, for example, such as intervals in a mud log. The data occurrences are assigned unique identifiers (UIDs) (see next section). WITSML randomly growing data-objects include trajectory and mud log. For more information, see Section 5.2, page 36.
Systematic	Data does not overlap, and the index of each row MUST be unique. For these data-objects, the equivalent of a table is defined and each update is essentially adding one row to the table. UIDs are not assigned to the data occurrences. WITSML currently has one systematically growing data-object, a log. For more information, see Section 5.3, page 37.

The specific details for special handling of growing data-objects are described in the section for the function to which they apply in Chapter 6, page 41.

2.1.1.2 NEW Data-object for 1.4.1: objectGroup

This release of WITSML includes a new data-object, which is named *objectGroup*.

Its main purpose is to provide a way to logically group logs that are part of a multi-pass/multi-run suite of logs or tests—logs that a user would only want to see as part of a collection.

In version 1.3.1, this functionality was included as part of the wellLog object. However, to streamline design and improve efficiency, the wellLog object has been deleted. All logs are defined with the log data-object. And the objectGroup is used to define a group of related logs.

The objectGroup has been designed so that it can be used to group any WITSML data-objects, if use cases determine the necessity.

For a complete definition of the data-object, see the data schema.

For a list of valid object group types (groupType) available in the current version, see the enumValues.xml file. In addition to these defined group types, a groupType may have a value of “other” (to provide more flexibility). If “other” is specified, then the user must also use the groupSubType element (free-form text) to specify the name of this other group. WITSML users are encouraged to submit new groupTypes to the SIG for future extension of the enumerated list.

For example queries of this data-object, see Section 14.2.6, page 139.

privateGroupOnly Element

Each data-object now includes in its commonData section, an element named *privateGroupOnly*, which is a Boolean flag used to indicate whether a data-object is part of a private group (that is, a group defined by the objectGroup data-object).

A client sets this flag. The default value is “false”. The flag should only be set to true when the data-object is part of a private group, for example, a log in a multi-pass/multi-run suite of logs or tests. If the value is “true”, when you query for data-objects (for example, logs), then you will NOT get these data-objects.

For example queries for grouped objects, see APPENDIX D: Examples, Section 14.2.6, page 139.

2.2 Object Identification: Unique Identifiers and Data-object Names

To identify, retrieve, and update WITSML data-objects, each data-object has a machine-readable unique identifier (UID) and a human-readable name element.

2.2.1 Unique Identifiers

Each WITSML data-object has within it a UID, and the identifier(s) of its ancestry. For instance, a mud log (mudLog) data-object has its uid attribute, its parent wellbore identifier attribute (uidWellbore), and its grandparent well identifier attribute (uidWell) (Figure 2).

The identifiers that represent parent data-objects are referred to as parentage pointers because they represent "foreign keys" that point to a UID in a parent data-object.

UIDs are typically assigned by client applications. However, in some cases, if the client does not define the data-object UID, then the server defines it and returns the value to the client. For more information, see Section 6.4, page 47.

If the generated uid already exists in the server, it is the client's responsibility to resolve.

Observe these guidelines for using UIDs:

- It is RECOMMENDED that the UID of each singular independent data-object be made globally unique by generating UIDs according to UUID algorithm of rfc 4122 (http://www.faqs.org/rfcs/rfc4122.html). An independent data-object is one that is not identified within the context of another data-object, for example, a well is an independent data-object.
- The unique identifier of each dependent data-object is REQUIRED to be unique within the context of its parent data-object. For example, the unique identifier in trajectory is unique within the context of its parent wellbore.
- A UID is constrained by the schema as follows: it cannot have any spaces and has a maximum length of 64 characters.
- Within a data-object, the identifier of all repeatable child elements is required to be unique with the context of the data-object, and the identifier of all descendant (child of child) repeatable elements is required to be unique within the context of the nearest repeatable element.
- The unique identification for any node is the logical combination of all UID values in the hierarchy down to and including that node (see Figure 2, page 13). Each UID value MAY actually be unique within a broader context but best practice SHOULD NOT assume a broader context.
- Each repeatable (i.e., more than one may occur within a parent) element that needs to be individually queried by the server (e.g., update a single node) must have a UID attribute. For example, within the well data-object, the wellDatum is a repeatable element.
- A non-repeatable element will not normally have a UID attribute. For the purposes of this discussion, a foreign key to a non-parent node is not considered to be a UID value. The only purpose of UID values is to insure uniqueness of nodes in a server. An example of a repeatable element is the wellDatum within the well data-object.
- Best practice SHOULD NOT assume any semantic content of a UID value.
- Requirements for using UIDs for data-objects vary across STORE functions. For more information, see Section 6.1.1, page 42.

2.2.2 Object Name Elements

Because system-oriented UIDs might contain internal keys or complex strings, elements have been defined to carry more "human readable" names of data-objects and their parentage, such as nameWell, nameWellbore, etc.

Any human-recognizable identifiers, such as name or index, for a node SHOULD be populated with values that identify the node within the context of the nearest repeatable parent node.

The natural key for a node is the combination of all natural identifiers in the hierarchy down to and including that node. While natural keys MAY be non-unique within the context of a server, every effort SHOULD be made to insure their uniqueness to eliminate ambiguity.

When the natural keys are not unique, then human inspection is needed to determine if the nodes represent:

- Same thing (i.e., if the information in the nodes SHOULD be combined) or
- Different things (i.e., their identity SHOULD be changed).

2.3 WITSML Document

Consistent with the way the XML literature uses the word "document," the term does not necessarily mean a physical file. A text string of XML being sent using an HTTP/S POST is also an XML document. If that XML represents one or more WITSML data-objects, it is then a WITSML document.

A WITSML document is one or more complete WITSML data-objects—such as well, wellbore or log—represented as an XML document, which is essentially a text string. That is, each WITSML data-object is defined by an XML schema and is its own document. Because each data-object carries with it its own identifier (UID) and the identifier(s) of its parent, grand-parent, etc, data items in one well or wellbore can be transported as one XML document (which again, is simply a text string).

So the data-objects can be logically thought of as being in one tree or hierarchy, but they are physically represented as separate documents. In Figure 2, page 13, that well and all of its child data-objects would be represented as thirteen WITSML documents, one for each data-object.

3. STORE Interface Overview

The WITSML API, named the STORE Interface, specifies a standardized way of electronically transporting WITSML data-objects between systems using the HTTP/S protocols—which includes HTTP (unencrypted) and HTTPS (encrypted), referred to collectively as HTTP/S.

The STORE Interface is platform independent and conforms to SOAP version 1.1.

The STORE Interface provides client applications the ability to interact with WITSML servers to:

- Add WITSML a data-object to a server.
- Update an existing WITSML data-object on a server.
- Delete an existing WITSML data-object (or a subset) on a server.
- Get (query) one or more existing WITSML data-objects from a server.

3.1 Client/Server Approach

A standardized Web Services Description Language (WSDL) file (see 4.3, page 34) is used to define the STORE Interface functions, which are exposed by SOAP. The function names (methods) are prefixed with "WMLS_" (a contraction of WITSML and STORE). The WITSML data-objects are exchanged between client and server as string values within XML documents.

3.1.1 Synchronous Transmission

When the client makes a request of a WITSML server, the server MUST respond immediately and synchronously with an acknowledgement or error code.

3.1.2 Representation vs. Transport vs. Storage

WITSML defines an interoperability specification about how the data is represented and a standard way of transporting the data electronically. It does not define the storage mechanism.

- **Representation.** WITSML data-objects being exchanged between systems are always represented as XML documents. The data-object schemas define the format of these data-objects when they are represented as XML documents.
- **Transport.** Because XML documents are simply text files, they can be transported by various means, such as email or other file transfer method. A WITSML data-object could even be in the form of a printed or faxed document.

  This API document specifies a standardized way of electronically transporting WITSML data-objects between systems using HTTP/S-based protocols. However, companies can send and receive WITSML data streams without using the STORE Interface.
- **Storage.** While WITSML data-objects transported between systems are represented as XML documents, the WITSML specification does not dictate how the WITSML data-objects are to be stored within those systems, and the client application cannot assume any particular storage mechanism. The data-objects might be stored as text files or in a database system.

3.2 Authentication and Encryption

The STORE interface uses HTTP/S Basic Authentication, as described in IETF RFC's 2616 (HTTP 1.1) and 2617 (Authentication).

The capabilities and implementation of authorization checking, if any, are not specified by WITSML. It is up to participating companies to agree if authentication is needed to access a WITSML server. If it is needed, the authentication data will be sent with each WITSML message.

Secure Sockets Layer (SSL) for HTTP (HTTPS) can be used to encrypt the HTTP traffic.

3.3 Server Data Compression

Servers MUST support gzip HTTP compression for server response and the compression must be compliant with IETF RFC 2616 (HTTP 1.1) Section 3.5 (Content Codings).

For client requests, some WITSML functions support alternate compression capabilities, which are explained in the sections for those functions in Chapter 6, page 41.

3.4 Key API Components

Key components of the STORE Interface are listed and described here.

Component	Description/Purpose
XML templates	The STORE interface uses the concept of a template, which is a well-formed XML document that specifies "by example" what WITSML data-objects and data items within those data-objects are to be acted upon. Clients use templates in the XMLin parameter of the various STORE interface functions. For more information on templates, see Section 4.1, page 23.
WSDL file	The WSDL file, which is sometimes called the API Signature, is a contract between the provider system (server) and the consumer system (client) that specifies how the Web service will interact in terms of what function calls are available, their parameters, the protocols they run over and where the service is located. The file is created using the Web Services Description Language (WSDL), an XML format for describing these types of network services. For more information, see Section 4.3, page 34.
Capabilities Objects	Special data-objects used by the STORE Interface to exchange information about the capabilities of a client or a server. The client and server each has its own Capabilities Object—capClient and capServer. For more information, see Section 4.2, page 28.
Functions	The basic tasks or functionality that can be performed with the STORE Interface are contained in functions. The functions are developed using WSDL. For more information, see Chapter 6, page 41.

3.4.1 Schema Variants

The following schemas represent generated variants of the normative data schemas. These schemas are only normative when used within the context of a Web service. They are used to accommodate the different requirements for mandatory data for the various actions (e.g., retrieve, add, update, and delete) of the Web services. The object plural *version* attribute is required in all variants.

Schema Type	Copies of the normative schema file with the following exceptions:
Read Schema	All elements and attributes are optional and all choices have been eliminated. If used within a WITSML Web service, these schema files must represent the XMLout response from the WITSML WMLS_GetFromStore function (see Section 6.6, page 54).
Write Schema	All uids and parentage-uids are mandatory (object-uids remain optional). If used within a WITSML Web service, these schema files must represent the XMLin input to the WITSML WMLS_AddToStore function (see Section 6.4, page 47).
Update Schema	All elements and attributes are optional except that all unique identifier attributes and uom attributes are mandatory. If used within a WITSML Web service, these schema files must represent the XMLin input to the WITSML WMLS_UpdateInStore function (see Section 6.6, page 54).
Delete Schema	All elements and attributes are optional except for all object uids and parentage-pointers, which are mandatory. If used within a WITSML Web service, these schema files must represent the QueryIn input to the WITSML WMLS_DeleteFromStore function (see Section 6.5, page 50).

3.5 Versioning

The STORE has four major components that can be versioned, each using versioning in slightly different contexts. The components and their corresponding version names are listed and summarized in the table below. Sections below provide more details for each component.

Note that the version numbering of these components is mostly independent of one another. The version numbers might increment at different rates and times for different parts of the WITSML specification.

Component	Version Name/Description
Data-objects	**Data Schema Version.** The version of the XML data schema against which the data-object can be validated. For more information, see Section 3.5.1, page 19.
WSDL file	**WSDL file version.** The version of the WSDL file is specified by the URI of the targetNamespace attribute of its <definitions> element. Once defined, the WSDL file SHOULD NOT change, else consumer applications that were written to an earlier specification might "break." For more information, see Section 3.5.2, page 20.
API Capabilities	Describes the details about how the client/server interface operates. Each time a client and server communicate, the capabilities version is exchanged through special client and server Capabilities Objects. Two versions are associated with API Capabilities: **API Version** is specified in the WITSML STORE application program interface (API) (this document). NOTE: The API version and the version of the API document are NOT the same thing. The API document can be updated (e.g., for clarification or error correction) that does not change API behavior. In this case, the API document version is updated, but not the API version. The API document version is found on page 2 and is formatted as “DOC1.4.1-v2”. **API Schema Version** is the version of the schema that defines the Capabilities Objects. For more information, see section 3.5.3, page 20.
Executable programs (either client or server applications)	The version of a particular implementation of all of the above. The version of an executable program can be used to help in troubleshooting. Each version may be coded to support only one combination of the above components. For more information, see section 3.5.5, page 22.

3.5.1 Data Schema Version

For a data-object, the Data Schema Version number specifies the version of the XML data schema against which the data-object can be validated. The Data Schema Version is conveyed in the version attribute of the data-object's root plural element:

```
<wells version="1.4.1.1" … >
      <well … />
</wells>
```

The example indicates that the well data-object is compliant with the WITSML 1.4.1.1 XML data schemas. A client can determine the Data Schema Versions of WITSML data supported by the server using WMLS_GetVersion function (see Section 6.2, page 44).

3.5.1.1 Significance of the Four Digits

Each of the four digits signifies a level of change. This numbering scheme applies to data schemas, the API, and API schemas.

- **First digit.** Represents a major structural or semantic change to all of the schemas as a whole. This is the level at which namespace changes are made.
- **Second digit**. Represents breaking changes across schemas, changes that would cause a document to have serialization or validation errors if read by a client or server at a previous revision in this digit.
- **Third digit**. Non-breaking additions to schemas only, which implies that only optional elements and attributes can be added to schemas and that elements and attributes could be marked "deprecated", but not removed from the schema.
  - This is the minimum level of change that is required to make a release of the schema package to the Energistics website.
  - It is intended that entirely new high-level data-objects (object_ schemas) could be added between major releases at the same version number of the current release (much like when the WITSML stimJob data-object was released), provided that the new data-objects do not introduce any breaking changes to included low-level schema components.
- **Fourth digit.** Changes in the XML files of standard enumerations and reference data. These documents would continue to validate at an XML level, but might encounter errors if presenting enumeration values that were unknown to the corresponding client or server.

3.5.2 WSDL File Version

The WSDL file, which is sometimes called the API Signature, is a contract between the provider system (server) and the consumer system (client) that specifies how the Web service will interact in terms of what function calls are available, their parameters, the protocols they run over, and where the service is located.

Once the WSDL is defined, it SHOULD NOT change, else consumer applications that were written to an earlier specification might "break." For more information on the WSDL file, see Section 4.3, page 34.

When new functionality needs to be added, the WSDL file can be updated but ideally in a way that extends but does not break existing functionality. Or if the changes are such that it is not possible to provide backward compatibility to existing clients, a completely new WSDL file (with a different Web service name) can be published.

The version of the WSDL file is specified by the URI of the targetNamespace attribute of its <definitions> element:

```
<definitions name="WMLS" targetNamespace="http://www.witsml.org/wsdl/120" …
```

A <documentation> element in the WSDL file also specifies the version in a more easily readable form:

```
<documentation>WITSML Version 1.2.0 STORE interface WSDL file </documentation>
```

Client programs SHOULD use the targetNamespace URI rather than the <documentation> element to determine the version of the WSDL file.

3.5.3 API Capabilities: API Version and API Schema Version

The API capabilities describe the "details" about how the client/server interface operates. An API's capabilities further define and clarify the basic functionality that the WSDL file advertises (think of the capabilities as being the "fine print" on the contract). For example, while the WSDL file says that a client can issue a WMLS_GetFromStore function call to retrieve WITSML data-objects from a server, it is the API's capabilities that determine which data-objects are available on a particular server.

The API capabilities version is referred to as the API Version.

The API Version is specified in the WITSML STORE application program interface (API) (i.e., this document). NOTE: The API Version and the version of the API document are NOT the same thing. The API document can be updated (e.g., for clarification or error correction) that does not change API behavior. The API document version is found on page 2 and is formatted as “DOC1.4.1-v2”.

Each API Version has a corresponding set of Capabilities Objects (one for a client and one for a server), which are a special kind of WITSML object. These objects are defined in an API schema which has an associated API Schema Version.

An API Version is associated with a single API Schema Version. However as the API specification is revised, a single API Schema Version can be used by multiple API Versions.

The API Version is conveyed in the apiVers attribute in the Capabilities Objects. The API Schema Version is included in the version attribute of the root element of the API objects (Capabilities Objects).

Here is an example of a server's Capabilities Object (capServer) showing the API Version and the API Schema Version:

```
<capServers ...version="1.4.1">        <!— API Schema Version ->
      <capServer apiVers="1.4.1">       <!— API Version ->
            …
      <capServer>
```

When a client initiates the WMLS_GetCap function, the server responds by sending its Capabilities Object (capServer). The client provides its Capabilities Object (capClient) in the CapabilitiesIn parameter for some STORE function calls.

Each Data Schema Version (see Section 3.5.4, page 22) has a single corresponding API Version. The API Version used by the server is based on the Data Schema Version provided by the client:

- The API Version sent in the server Capabilities Object in the response to a WMLS_GetCap function is based on the dataVersion sent in the request
- The API Version use by the server for the WMLS_AddToStore, WMLS_GetFromStore, WMLS_DeleteFromStore, and WMLS_UpdateInStore functions is based on the Data Schema Version in the version attribute in the QueryIn parameter.
- For more information about Capabilities Objects, see Section 4.2, page 28.
- For more information on the WMLS_GetCap function, see Section 6.3, page 45.

3.5.4 Relationship among these Versions Types

Figure 3 shows how versions of different WITSML components are related. An API can work with different API Versions and API Schema Versions. For example, if you added a new data-object to a schema, it would not (necessarily) require a change to the API.

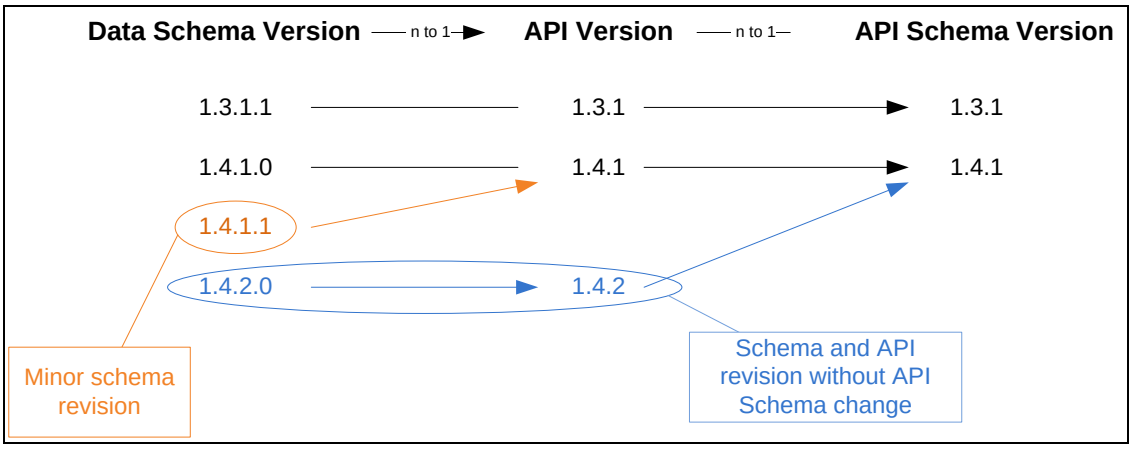


Figure 3 Relationship among the versions of key WITSML components. The information in blue is hypothetical, for purpose of example to show how different versions of components could interact.

3.5.5 Versioning: Executable Programs

The version or "build" number of an executable server program is provided as additional information, and can be used to assist in troubleshooting, for example. The program's version/build number is chosen solely by the particular implementer and does not necessarily correlate to the version numbering scheme used by the other WITSML entities.

Here is an example of a server's CapabilitiesObject (capServer) showing the executable version:

```
<capServers ...>
      <capServer>
            <version>1.4.1.1463</version>
      <capServer>
</capServers>
```

4. Templates, Capabilities Objects, and WSDL File

This chapter covers details of some main API components including:

- XML templates
- Capabilities Objects (for client and server)
- WSDL file

NOTE: The data schema files specify the normative definition of the XML data-objects and structures. This API document defines normative behavior associate with those objects. If the API and a schema differ on the name of an element or attribute, the schema defines the normative name.

The other main API components—functions—are discussed in Chapter 6, page 41.

4.1 XML Templates

This section introduces the behavior of the XML template used by the STORE interface. Each function expands on this concept, providing details for specific functions.

4.1.1 Introduction

The STORE interface uses the concept of a template, which is a well-formed XML document that specifies by example what WITSML data-objects and data items within those data-objects are to be acted upon.

the Query template

```
<wells xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
       <well uid="123">
             <name/>
       </well>
</wells>
```

the query

This query template (used on the WMLS_GetFromStore function, page 54) contains only one query, which requests the name of the well data-object that has an ID of “123”.

4.1.1.1 Template Rules

- All client query templates MUST **[else error -401]** contain a plural root element; in the example above, <wells>.
- The plural root element can contain one or more singular elements of the same data-object type, in this case <well>; however, most functions constrain the template to have only one singular element. Each <well> element in this example query template is a separate query.

```
<wells xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
   <well uid="123">
         <name/>
   </well>
   <well uid="456">
         <name/>
   </well>
</wells>
```

1st query

2nd query

- Query templates are intended to provide a simple means of manipulating data-objects based on an equal comparison against a single value for each data item, either attribute or element. They do not provide the ability to perform comparisons other than “equal” nor do templates allow for more than one comparison against a given data item.

  However, though the default comparison is “equal,” a few date and time elements in the WITSML specification use “greater than” comparisons. These differences are documented in this API specification.

- The “1series” in the name space element refers to the numbering scheme for WITSML versions. For more information on the digits in the version number, see Section 3.5.1.1, page 20.

Version Attribute Rules

- A client MUST **[else error -468]** include the version attribute in the plural object defining the Data Schema Version for all templates.
- The server MUST return the version attribute in the plural object defining the Data Schema Version.
- The value of the version attribute must match one of the Data Schema Versions on the list returned by WMLS_GetVersion (see Section 6.2, page 44). This applies to client in and server out.

4.1.2 Data Item Selection

To specify the data items—either attributes or elements—to be returned by a query, the client query includes the XML attribute(s) and elements(s) ***without*** an associated value.

For…	Specify…
XML attributes	`attribute=""`
XML elements	`<element></element> or simply <element/>`

There is no syntax within a query template for returning all data items, other than specifying all the attributes/elements individually. However, the query parameter “returnElements=all” achieves the same result.

For data item selection, if a server supports a data-object, then it MUST support all the items (elements or attributes) for that data-object as specified in the data schema.

Support of an item for data item selection does not guarantee that a value for the corresponding item will be included in the response, because some items may not have been populated in the server. However, all mandatory items in the write schema will be guaranteed to be populated with values in the response.

```
<wells xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
       <well uid="">
              <name/>
              <country/>
       </well>
</wells>
```

…and the server returns:

```
<wells xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
       <well uid="aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa">
              <name>HLH 4 Walker Survey</name> <!—- Mandatory element. Always has a value
in response -->
              <country>Canada</country> <!—- Optional element ‘country’ has value for this
object -->
       </well>
       <well uid="aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaab">
              <name>Watson 6 Sewell Survey</name> <!—- Mandatory element. Always has
a value in response -->
              <!—- Optional element ‘country’ does not have value for this object -->
       </well>
       <!-- and so on for all well data-objects on the server -->
</wells>
```

4.1.3 Data-object Selection

To specify the data-objects to be returned by a query, the client query template includes the attribute(s) or element(s) in the query template ***with*** an associated value.

The server MUST return the data-object selectors in addition to the data item selectors.

For…	Specify…
XML attributes	`attribute="value"`
XML elements	`<element>value</element>`

Example: The following query includes a value for the `name` element:

```
<wells xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
      <well>
            <name>example 6</name>
      </well>
</wells>
```

In response to this query, the server returns all data-objects that match the criteria. However, what we get back will match the query we specified, so the result will not be very useful. For a more useful result, see Section 4.1.4.

A server MAY only support a subset of elements and attributes for data-object selection. However, if a server supports a data-object, then it MUST support all the data-object selection items specified for that data-object in Chapter 7, page 97.

4.1.3.1 How a Client Determines which Elements a Server Supports for Data-object Selection

To determine which elements and attributes a server supports for data-object selection, a client can:

- Call the WMLS_GetFromStore function, and in the OptionsIn parameter set the keyword requestObjectSelectionCapability=true (see Section 6.6.2.1, page 55).

In response to this request, the server MUST return a single object of the type requested, containing only the elements and attributes that it supports for object selection. The values for the elements and attributes to be used are specified in APPENDIX B: Server Object Selection Capability Templates, page 114.

Example: A server that supports the *country* and *operator* elements (in addition to the mandatory elements and attributes) for well object selection will return the following object when it receives a WMLS_GetFromStore call for a well with requestObjectSelectionCapability=true**.**

For more information on using requestObjectSelectionCapability (and other OptionsIn parameters), see page Section 6.6.2.1, page 55.

```
<wells xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
      <well uid="abc">
            <name>abc</name>
            <numGovt>abc</numGovt>
            <country>abc</country>
            <operator>abc</operator>
            <numAPI>abc</numAPI>
            <commonData>
                  <dTimLastChange>1900-01-01T00:00:00.000Z</dTimLastChange>
            </commonData>
      </well>
</wells>
```

4.1.4 Combining Data Item and Data-object Selection

To get more useful data-objects, a query template can also combine data-object selection with data item selection.

Example: To retrieve the unique identifier associate with the well with a specified name, you can specify:

```
<wells xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
      <well uid="">
            <name>example 6</name>
      </well>
</wells>
```

In this example, name is a data-object selection criterion (because it has a value) and uid is an item selection criteria, because it does not have a value.

The server returns a well data-object for each well on the server that has the specified name (hopefully only one). The server also returns the name data item and the well's unique identifier (UID).

4.1.4.1 Multiple Data-object Selection Criteria

A query can also specify multiple data-object selection criteria.

- When a query contains multiple data-object selection criteria, the server MUST "AND" the criteria together. That is, the server MUST return only the data-objects where ALL data-object selection criteria are true.

Example: To return all logs for a particular wellbore, specify two data-object selection criteria:

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
      <log uidWell="w1" uidWellbore="wb1" uid="">
            …
      </log>
</logs>
```

4.1.5 Selection using Recurring Data Items

- When a query includes recurring data, the server MUST "OR" the criteria.

Example: For the following query, the server returns all curve descriptions where: `(uidWell=w1 AND uidWellbore=w1b AND uid=log99 AND(mnemonic=BPOS OR mnemonic=ROP)).`

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
      <log uidWell="w1" uidWellbore="w1b" uid="log99">
            <logCurveInfo>
                  <mnemonic>BPOS</mnemonic>
            </logCurveInfo>
            <logCurveInfo>
                  <mnemonic>ROP</mnemonic>
            </logCurveInfo>
      </log>
</logs>
```

- When a query includes multiple selection criteria in the recurring element, then the server MUST "AND" the criteria first, before "ORing" the criteria.

Example: For the following query, the server returns all changeLog entries where:

```
(uidWell=w1 AND uidWellbore=w1b AND uidObject=f34a AND
 ((dTimChange>2010-11-24T08:15:00.000Z AND changeType=update) OR
  (dTimChange>2010-11-24T08:15:00.000Z AND changeType=delete)
 )
).
```

This query assumes that "returnElements=all" in the OptionsIn parameter. Note that dTimChange is an exception in WITSML that uses a "greater than" comparison instead of "equal" (For more information, see Chapter 9, page 101.).

```
<changeLogs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
      <changeLog uidWell="w1" uidWellbore="wb1" uidObject="f34a">
            <objectType>log</objectType>
            <changeHistory>
                  <dTimChange>2010-11-24T08:15:00.000Z</dTimChange>
                  <changeType>update</changeType>
            </changeHistory>
            <changeHistory>
                  <dTimChange>2010-11-24T08:15:00.000Z</dTimChange>
                  <changeType>delete</changeType>
            </changeHistory>
      </changeLog>
</changeLogs>
```

- The query MUST **[else error -438]** specify the same selection items in each recurring node.

Example: The following template requests `mnemonic` in one `logCurveInfo` item and `unit` in another.

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
      <log uidWell="w1" uidWellbore="w1b" uid="log99">
            <logCurveInfo>
                  <mnemonic>BPOS</mnemonic>
            </logCurveInfo>
            <logCurveInfo>
                  <unit>ft</unit>
            </logCurveInfo>
      </log>
</logs>
```

Error because the recurring items are not symmetrical.

- The query MUST NOT **[else error -439]** specify selection criteria containing multiple recurring items that contain an empty value.

Example: The following template requests `mnemonic` BPOS both explicitly (using `<mnemonic>BPOS<mnemonic>`) and implicitly (using `<mnemonic/>`).

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
      <log uidWell="w1" uidWellbore="w1b" uid="log99">
            <logCurveInfo>
                  <mnemonic>BPOS</mnemonic>
            </logCurveInfo>
            <logCurveInfo>
                  <mnemonic/>
            </logCurveInfo>
      </log>
</logs>
```

Error because an empty recurring value is not allowed.

4.1.6 Minimum Query Template

A “query template” is an alias for the QueryIn parameter on the WMLS_GetFromStore function. For more information, see Section 6.6.2, page 54. For a list of WITSML standard query templates, see Section 6.6.5.

- A query template MUST **[else error -469]** be a well formed XML document that includes only attributes and elements defined by the data schema for the particular data-object type.
- A query template MUST **[else error -403]** include a default namespace declaration for the WITSML namespace (see below), and any other necessary namespace.

- However, a query template need not be a valid WITSML document, because empty values are not valid for some elements and some functions require use of empty values.

The examples in this specification use the following namespace from the data schema that was current at the time this specification was published. Use of these examples with a different data schema might require a different namespace.

```
xmlns=http://www.witsml.org/schemas/1series
```

Therefore, the minimum valid query template to return all the wells on a server is:

```
<wells xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
       <well/>
</wells>
```

For real queries against a server, the namespace of the actual data schema must be substituted for the example data schema namespace.

The above example returns only the following—nothing else (assuming that “returnElements = all” was not used in the OptionsIn keyword; see Section 6.6.2.1, page 55).

```
<wells xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
       <well/>
       <well/>
       …
</wells>
```

This information is sufficient if all you need is a count of all the wells on the server. However, if you need additional data items about each well, you must ask for them by including the various elements/attributes in the query template.

4.2 Capabilities Objects

To exchange information about the capabilities of a client and a server, the STORE interface uses two special data-objects called Capabilities Objects. WITSML defines Capabilities Objects for a client and a server.

NOTE: The WITSML schema files specify the names and order of the elements and attributes (items). This API document defines and specifies the behavior of these items. For discrepancies discovered between the two sources, the sources as defined are the definitive authorities (that is, schema for item names and API document for item definitions and behaviors). Please report any such discrepancies to Energistics using the contact form on our website http://www.energistics.org/contact-us.

4.2.1 Restriction

The Capabilities Objects cannot be accessed by the STORE interface’s standard data query functions. The Capabilities Objects are not persisted as data. They can only be accessed using the WMLS_GetCap function (see page45) and the Capabilities In/Out parameters of the various STORE interface functions.

4.2.2 Identifying a Client Using the HTTP User-Agent

A client MUST **[else error -472]** identify itself to a server using the HTTP user-agent field. The client must pass its product name (preferably the short name) and version number as defined in the W3C standard (http://www.w3.org/Protocols/HTTP/HTRQ_Headers.html#user-agent).

This agent is typically used by HTTP to direct behavior, for example, for different browsers. However, the agent is being used by WITSML to identify clients to help with troubleshooting when problems occur with client/server interaction.

4.2.3 Client Capabilities (capClient) Object

- A client application uses its Capabilities Object or capClient object to inform a server of its functional capabilities.
- The capClient object is conveyed as the CapabilitiesIn parameter of various STORE interface functions. If a client passes a capClient object, it SHOULD pass it on each call because the server does not maintain the value between calls.
- The client MAY set the CapabilitiesIn parameter to an empty string to indicate that the client is not providing a capClient object.
- It is RECOMMENDED that this parameter be implemented under user or site control. For complicated scenarios, this information can be very useful for enabling site diagnostics. However, for some low-bandwidth scenarios, passing the extra information can dramatically slow performance.
- To determine which Data Schema Versions (and therefore which API Versions) that the server supports, a client should use the WMLS_GetVersion function (page 54). After the client receives the server response, the client MUST **[else error -467]** then pass an API Version that the server supports.

4.2.3.1 capClient Attributes and Elements

The following table lists and describes the attributes and elements of the capClient object.

Attribute/Element	Description/Purpose/Example

apiVers	Required
	Specifies the API Version that the client supports. This API Version MUST **[else error -465]** match one of the API Schema Versions used. For example:
	`<capClients xmlns="http://www.witsml.org/api/141" version="1.4.1"`
	`<capClient apiVers="1.4.1">`
	
	`</capClient>`
	`</capClients>`

contact	Optional
	Provides contact information for the client by way of the child elements name, email and phone. There is no default value. For example:
	`<capClient …>`
	`  <contact>`
	`  <name>Bob Junior</name>`
	`  <email>bobj@aol.com</email>`
	`  <phone>1-800-555-1212</phone>`
	`  </contact>`
	`</capClient>`

description	Optional
	Provides a textual description of the client. There is no default value. For example:
	`<capClient …>`
	`  <description>CLIENT SOFTWARE A</description>`
	`</capClient>`

name	Optional
	Provides a meaningful name for the client. There is no default value. For example:
	`<capClient …>`
	`  <name>client #1</name>`
	`</capClient>`

vendor	Optional Identifies the vendor (producer) of the client software. There is no default value. For

Attribute/Element	Description/Purpose/Example
	example: `<capClient …> <vendor>Acme Downhole Software</vendor> </capClient>`
version	Optional Defines the build number of the client executable program (See Section 3.5.5, page 22). There is no default value. For example: `<capClient …> <version>1.4.1.1451</version> </capClient>`
schemaVersion	Required Defines all of the Data Schema Version(s) (for definition, see Section 3.5.1, page 19) that the client supports. The values MUST **[else error -473]** match the version attribute used in the plural data-objects. The value is a comma separated list of values without spaces. The client MUST **[else error -404]** order the list oldest to newest Data Schema Versions supported. This ordering is required so that newer Data Schema Versions will not confuse older servers. Example: `<capClient …> <schemaVersion>1.2.0,1.3.1.1,1.4.1.1</schemaVersion> </capClient>`

4.2.3.2 capClient Object Example

The following is example XML code of a client Capabilities Object using all attributes and elements described above. **Reminder:** All information contained here is about the client software; the client is providing this information to the server.

```
<capClients
      xmlns="http://www.witsml.org/api/141"
      version="1.4.1">                              <!– API Schema Version ->
      <capClient apiVers="1.4.1">                     <!– API Version ->
            <contact>
                  <name>John Smith</name>
                  <email>myEmailAddress@abc.com</email>
                  <phone>888-123-4567</phone>
            </contact>
            <description>Research dept at ABC corp</description>
            <name>Client #1 at ABC corp.</name>
            <vendor>Acme Downhole Software</vendor>
            <version>1.4.1.1451</version>        <!– Executable program version ->
            <schemaVersion>1.3.1.0,1.4.1.1</schemaVersion>   <!– Data Schema
Versions ->
      </capClient>
</capClients>
```

4.2.4 Server Capabilities (capServer) Object

The server Capabilities Object or capServer object is used to inform the client of the server's functional capabilities. The capServer object is conveyed as the CapabilitiesOut parameter of the WMLS_GetCap function (page 45).

4.2.4.1 capServer Attributes and Elements

The following table lists and describes the attributes and elements that make up the capServer object. For a code example of a complete capServer object that uses each of these elements, see Section 4.2.4.2, page 34.

Attribute/Element	Description/Purpose/Example

apiVers	Required
	Specifies the API Version that the server supports, which is associated with the Data Schema Version provided in the dataVersion element of the WMSL_GetCap Function. This API Version MUST match one of the API Versions associated with the API Schema Version used (For definitions and information about versions, see Section 3.5, page 19). For example:
	`< capServers  xmlns="http://www.witsml.org/api/141" version="1.4.1"`
	`  <capServer apiVers="1.4.1">`
	`  …`
	`  </capServer>`
	`</capServers>`

contact	Optional
	Provides contact information for the server by way of the child elements name, email and phone. For example:
	`<capServer …>`
	`  <contact>`
	`  <name>Bob Junior</name>`
	`  <email>bobj@aol.com</email>`
	`  <phone>1-800-555-1212</phone>`
	`  </contact>`
	`</capServer>`

description	Optional
	Provides a textual description of the server. For example:
	`<capServer …>`
	`  <description>Equipment Room A - Rack 12</description>`
	`</capServer>`

name	Optional
	Provides a meaningful name for the server. For example:
	`<capServer …>`
	`  <name>server #1</name>`
	`</capServer>`

vendor	Optional
	Identifies the vendor (producer) of the server software. For example:
	`<capServer …>`
	`  <vendor>Acme Server Software Company</vendor>`
	`</capServer>`

version	Optional
	Defines the build number of the server executable program (See Section 3.5.5, page 22). For example:
	`<capServer …>`
	`  <version>1.4.1.1451</version>`
	`</capServer>`

schemaVersion	Required
	Defines the Data Schema Version for which the server capabilities apply (i.e. the Data Schema Version provided in the dataVersion of the WMLS_GetCap function). The server MUST ensure that the values match the version attribute used in the plural data objects. For example:
	`<capServer …>`
	`  <schemaVersion>1.4.1.1</schemaVersion>`
	`</capServer>`

<table>
<tr><th>Attribute/Element</th><th>Description/Purpose/Example</th></tr>
<tr><td>changeDetectionPeriod</td><td>Optional. Required for servers that support growing data-objects. Maximum value = 600 seconds (10 minutes)
Defines the maximum number of seconds for the server to detect change in a growing data-object. A server MUST detect a change to any growing data-object within the specified time.
For more information on how this is used, see Chapter 9, page 101. For example:
<pre><capServer …>
  <changeDetectionPeriod>36</changeDetectionPeriod>
</capServer></pre></td></tr>
<tr><td>maxRequestLatestValues</td><td>Required. Minimum = 1
For a log data-object, defines the maximum number of values per curve that the server returns when the client requests the latest values. The value is a positive integer with a minimum value of 1. Even if a client requests more values, the server only returns whatever its maxRequestLatestValues is.</td></tr>
<tr><td>growingTimeoutPeriod</td><td>Optional. Required for servers that support growing data-objects.
Defines the maximum number of seconds that a server will consider a growing data-object to be growing when no data is being appended to the object. A separate time is defined for each growing data-object type. If data is not appended within the specified time, the server sets the objectGrowing flag for the data-object to “false”. A server MUST define a value for each supported growing data-object type.
For more information on how this is used, see Chapter 9, page 101. For example:
<pre><capServer …>
  <growingTimeoutPeriod dataObject="log">60</growingTimeoutPeriod>
</capServer></pre></td></tr>
<tr><td>cascadedDelete</td><td>Optional
Indicates whether the server supports the cascadedDelete OptionsIn keyword. True indicates that the server supports the cascadedDelete OptionsIn keyword. False or not given indicates the server does not support cascaded deletes. For example:
<pre><capServer …>
  <cascadedDelete>false</cascadedDelete>
</capServer></pre></td></tr>
<tr><td>compressionMethod</td><td>Optional
Defines a comma delimited list of compression methods that a server supports for requests. On the response, all WITSML servers MUST support gzip HTTP compression.
Not given or a singular value of "none" indicates that the server does not support compression. For example:
<pre><capServer …>
  <compressionMethod>gzip</compressionMethod>
</capServer></pre>Currently recognized compression methods include:
<table><tr><th>Value</th><th>Definition</th></tr><tr><td>None (DEFAULT)</td><td>Specifies that the server does not support compression for client requests (QueryIn).</td></tr><tr><td>gzip</td><td>Specifies that the server supports gzip compression for client requests.</td></tr></table></td></tr>
</table>

Attribute/Element	Description/Purpose/Example

function	Optional. Not required if the server supports no additional optional functions. (For list of functions, see page 41.)
	Recurring element defines the STORE functions that the server supports and defines the versions, data-objects, and maximum size constraints that the server supports for each function.
	See additional attributes and elements listed below pertaining to the function element.
	`<capServer …`
	`  <function name="WMLS_GetFromStore">`
	`  <dataObject>well</dataObject>`
	`  <dataObject>wellbore</dataObject>`
	`  <dataObject>message</dataObject>`
	`  <dataObject maxDataNodes="1000"`
	`  maxDataPoints="50000">log</dataObject>`
	`  </function>`
	`  <function name="WMLS_AddToStore">`
	`  <dataObject>message</dataObject>`
	`  </function>`
	`</capServer>`

name (of function element)	Required for each function element that is specified. Specifies the name of the optional API Store functions that a server supports and may include: WMLS_AddToStore, WMLS_DeleteFromStore, WMLS_GetBaseMsg, WMLS_GetFromStore, or WMLS_UpdateInStore. **NOTE:** All WITSML servers MUST support WMLS_GetVersion and WMLS_GetCap function, so they need not be listed.
dataObject (of function element)	Required for each function element that is specified. Defines which data-objects are supported for the specified function. For information about items and attributes that must be supported for data-object selection, see Chapter 7, page 97.
maxDataNodes (of dataObject element)	Required only for growing data-objects (see Chapter 5, page 36). Represents the maximum number of data-nodes that can be handled by the specified function for each specified growing data-object.
maxDataPoints (of dataObject element)	Required only for systematically growing data-objects (see Section 5.3, page 37). Represents the maximum number of columns times the number of rows that can be handled by the specified function for each specified systematically growing data-object.
supportUomConversion	Required. Boolean. If server supports unit of measure (uom) conversion, set to “true”. If not, “false”. Default is false. For more information about units of measure conversion, see Section 8.4, Item 1, page 100.

4.2.4.2 Example

This is example code for a server Capabilities Object that uses all attributes and elements listed above.

```
<capServers xmlns="http://www.witsml.org/api/141" version="1.4.1">
	<capServer apiVers="1.4.1">
		<contact>
			<name>server contact name</name>
			<email>serverAdministrator@serverCompany.com</email>
			<phone>281-135-7924</phone>
		</contact>
		<description>server description</description>
		<name>John Smith</name>
		<vendor>INSITE server #1</vendor>
		<version>1.1.0.120</version>
		<schemaVersion>1.4.1.1</schemaVersion>
		<changeDetectionPeriod>36</changeDetectionPeriod>
		<growingTimeoutPeriod dataObject="log">60</growingTimeoutPeriod>
		<growingTimeoutPeriod dataObject="mudLog">60</growingTimeoutPeriod>
		<cascadedDelete>false</cascadedDelete>
		<compressionMethod>gzip</compressionMethod>
		<profileName>My Company Log Standard</profileName>
		<function name="WMLS_GetFromStore">
			<dataObject>well</dataObject>
			<dataObject>wellbore</dataObject>
			<dataObject maxDataNodes="1000" maxDataPoints="10000">log</dataObject>
			<dataObject maxDataNodes="200">mudLog</dataObject>
		</function>
		<function name="WMLS_AddToStore">
			<dataObject maxDataNodes="500" maxDataPoints="5000">log</dataObject>
			<dataObject maxDataNodes="200">mudLog</dataObject>
		</function>
		<function name="WMLS_UpdateInStore">
			<dataObject maxDataNodes="500" maxDataPoints="5000">log</dataObject>
			<dataObject maxDataNodes="200">mudLog</dataObject>
		</function>
		<function name="WMLS_GetBaseMsg"/>
	</capServer>
</capServers>
```

4.3 WSDL File

The Web Service Description Language (WSDL) file is used to expose the STORE interface to SOAP clients. Developers use the WSDL file to create proxies for their applications. This section provides an overview of key points about the WITSML WSDL file.

NOTE: To ensure interoperability between implementations, the only part of the WSDL file that should be modified is the location attribute of the SOAP address element.

The WSDL file is distributed as part of the WITSML standard, which can be downloaded from http://www.energistics.org/drilling-completions-interventions/witsml-standards/current-standards, and provides normative schema element and attribute names.

For an easy-to-read, understandable introduction to WSDL files, see the article titled “Web Services Description Language (WSDL) Explained” by Carlos C. Tapang - Infotects (in the Microsoft MSDN Library and elsewhere).

4.3.1 Key Points about the WITSML WSDL File

- The only part of the WSDL file that SHOULD normally be modified is the location attribute of the SOAP address element. This SHOULD be set to the URL of the Web Service (see implementation specifics below in this section):

```
<service name='WMLS'
   <port name='StoreSoapPort' binding='wsdlns:StoreSoapBinding' >
   <soap:address location='http://yourorg.com/yourwebservice' />
   </port>
</service>
```

- The WSDL file complies with Version 1.2 of the Web Services Description Language (http://www.w3.org/TR/2003/WD-wsdl12-20030303/)
- A client cannot assume that a server will provide a WSDL file.
- The target namespace and wsdlns prefix used in the WSDL file are both declared as:

```
            'http://www.witsml.org/wsdl/120'
   <definitions name='WMLS'
         targetNamespace='http://www.witsml.org/wsdl/120'
         xmlns:wsdlns='http://www.witsml.org/wsdl/120'
```

- A default namespace declaration is used to make the document more readable:

```
   <definitions name='WMLS' targetNamespace='http://www.witsml.org/wsdl/120'
         xmlns:wsdlns='http://www.witsml.org/wsdl/120'
         xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
         xmlns:xsd='http://www.w3.org/2001/XMLSchema'
         xmlns='http://schemas.xmlsoap.org/wsdl/'>
```

- The soapAction attribute of the SOAP operation elements specify a WITSML-related URI: http://www.witsml.org/action/120/Store.WMLS_xxxx

```
<operation name='WMLS_AddToStore'>
   <soap:operation
soapAction='http://www.witsml.org/action/120/Store.WMLS_AddToStore'/>
<input>
```

- The namespace attribute of the SOAP body elements also specifies a WITSML-related URI: http://www.witsml.org/message/120.

```
    <operation name='WMLS_AddToStore' >
      <soap:operation
          soapAction='http://www.witsml.org/action/120/Store.WMLS_AddToStore' />
      <input>
         <soap:body use='encoded' namespace='http://www.witsml.org/message/120'
           encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
      </input>
    </operation>
   …
```

- Although XML generally ignores white space, the values of certain XML attributes and elements might be sensitive to the amount of white space. For example, the parameterOrder attribute of the <operation> element can have only one space between parameters:

```
<operation name='WMLS_AddToStore'
   parameterOrder='WMLtypeIn XMLin OptionsIn CapabilitiesIn SuppMsgOut'>
```

5. Growing Data-objects

To meet the needs of drilling operations, WITSML defines growing data-objects, which, in some cases, must be handled differently than standard WITSML data-objects.

- This chapter defines growing data-objects, related concepts for handling them, and common behavior.
- The specific details for special handling growing data-objects are described in the section for the function to which they apply in Chapter 6, page 41.
- For a list of and examples of standard WITSML queries, see Section 6.6.5, page 65.
- For additional examples for growing data-objects, see Appendix D, Section 14.2, page 122.

5.1 What is a Growing Data-object and why is it Different?

Drilling wells results in data-objects that grow over time and as such have been named growing data-objects, which are characterized by:

- Relatively large number of occurrences of recurring data that is indexed to time or depth.
- Additional occurrences of recurring data that are inserted (or updated) over time (which is what causes the data-object to grow).

To accommodate the need to selectively retrieve only a range of occurrences within a structure of a growing data-object, WITSML defines special handling rules, which modify the standard behavior of query templates.

WITSML defines two types of growing data-objects:

Type of Growing data-object	Definition
Random	• Data-objects are unrelated and might overlap or coexist. • WITSML randomly growing data-objects include trajectory and mud log. • These data-objects are assigned UIDs. For a detailed definition, see Section 5.2, page 36.
Systematic	• Data does not overlap. • For these data-objects, the equivalent of a table is defined and each update is adding one or more rows to the table. UIDs are not assigned. • WITSML systematically growing data-objects include a log. For a detailed definition, see Section 5.3, page 37.

5.2 Randomly Growing Data-objects (trajectory and mudLog): Definitions and Concepts

This section defines a randomly growing data-object and other important data elements as defined and documented in the WITSML schema. Concepts explained here are mapped to the data schema in Section 5.4, page 40. Servers MUST support and understand randomly growing data-objects, as described in specific function sections of Chapter 6.

- **Structural-range**. In a non-recurring portion of the data-object (e.g., in a header) there is one or more pairs of elements that each represent a range of index values. Each pair represents a measure, a coordinate, or a date-time value. These elements represent the range (i.e., minimum and maximum) of index values that exist in the substructures. That is, they represent a structural-range.

Each individual element might represent concepts such as start, end, minimum, maximum, top or bottom, but the content of the pair must represent a minimum and maximum. For example, the following represent structural-range elements:

- startMd and endMD in mudLogs,
- mdMn and mdMx in trajectory

Regardless of their standard definitions, for the purpose of being a query parameter, their meanings are a minimum and a maximum with no implication of a sorting order (i.e., sorting only applies to the result).

- **Node-index and data-node.** Each recurring substructure contains element(s) representing a particular index point or index interval that locates each node in the index space. These indexes represent the node-index. This substructure represents the data-nodes.
  - Example node-indexes are: mdTop and mdBottom in the mudLog's parameter and geologyInterval, md in trajectoryStation.
  - Example data-nodes are: (mudLog) parameter, geologyInterval, and trajectoryStation.
- The recurring substructure(s) have a uid attribute to uniquely identify each node.

5.3 Systematically Growing Data-objects (logs): Definitions and Concepts

This section describes how a systematically growing data-object is defined and documented in the WITSML schema. Currently, a log is the only systematically growing-data object defined in WITSML. Concepts explained here are mapped to the data schema in Section 5.4, page 40. Servers MUST support and understand systematically growing data-objects, as described in specific function sections of Chapter 6.

- The data-object represents a table or a set of tables.
- **Structural-range.** In a non-recurring portion of the data-object (e.g., in a header) there are one or more pairs of elements that each represent a range of index values. Each pair represents a measure, a coordinate or a date-time value. These elements represent the range (i.e., minimum and maximum) of index values that exist in the data-object. That is, they represent a structural-range.

  Each individual element might represent concepts such as start, end, minimum, maximum, top or bottom but the content of the pair represents a minimum and maximum. For example, startIndex and endIndex in the log represent structural-range elements.
- **Node-index and data-node.** Each table is assigned a recurring structure that represents a row in that table. This structure represents the data-nodes (i.e., rows). This recurring structure does not have a uid attribute. The recurring row structure contains a point value (i.e., not a pair representing an interval) in its indexing space. This value represents the node-index. Note that this value might be implicitly defined (e.g., using start and increment) or be explicitly defined as a column. An example node-index is the data value representing the index curve in log data. An example data node is data in logData.

  NOTE: A systematically growing data-object cannot contain multiple nodes with the same index.
- **Column-identifier and column-range.** Each table definition structure is assigned a recurring column-definition structure to describe each column in the table. Each column has a contextual column-identifier (separate from a uid). For example, mnemonic in logCurveInfo represents a column-identifier.

  For manipulating data-nodes, the client MUST **[else error -449]** specify the column-identifier within a data-column-list. If a client specifies a column-identifier representing the index column, then the client MUST **[else error -457]** specify it first in the data-column-list.

  Each column has one or more pairs of elements that represent a column-range of index values for that column. The column-range values represent the range where values of that column actually

exist in the table. Null or empty values are excluded from this range. The range start is the first occurrence of a non-null value, and the range end is the last occurrence of a non-null value.

Note: Occurrences of null or empty values in the middle of the data set should have no affect on the column-range values.

5.3.1 Grouped Logs

In the current API version of WITSML, functionality that was previously included in wellLog data-objects has been replaced with a new data-object called *objectGroup*.

The objectGroup provides a way to logically group log data-objects to that are part of a multi-pass/multi-run suite of logs or tests—logs that a user would only want to see as part of a collection.

The objectGroup has been designed so that it can be used to group any of the WITSML data-objects, if use cases determine the necessity.

For a complete definition of the data-object, see the data schema.

For more information about elements and attributes that must be supported for object selection, see Chapter 7, page 97.

privateGroupOnly Element

Each data-object now includes in its commonData section, an element named *privateGroupOnly*, which is a Boolean flag used to indicate whether a data-object is part of a private group (that is, a group defined by the objectGroup data-object).

A client sets this flag. The default value is “false”. The flag should only be set to true when the data-object is part of a private group, for example, a log in a multi-pass/multi-run suite of logs or tests. If the value is “true”, when you query for data-objects (for example, logs), then you will NOT get these data-objects.

For example queries for grouped objects, see APPENDIX D: Examples, Section 14.2.6, page 139.

5.3.2 Common STORE API Behavior for Logs

This section describes STORE interface behavior common to all systematically growing data-objects (currently, a log) as defined by WITSML. The special handling required for each individual STORE function is explained in Chapter 6, page 41.

For systematically growing data-objects:

- The column-identifier values are case insensitive, the same as unique identifier (UID) values.
- When adding or updating curves, a client MUST **[else error -447]** specify unique column-identifiers.
- A server MUST return unique column-identifiers.
- The column-identifier values MUST NOT **[else error -459]** contain any of the following special characters:
  - single-quote (‘)
  - double-quote (“)
  - less than (<)
  - greater than (>)
  - forward slash, (/)
  - backward slash (\)
  - ampersand (&)
  - comma (,)

- The structural-range and column-range elements are read-only elements; the server MUST ignore any request to change them without issuing an error.
- For behavior related to units of measure, see Chapter 8, page 99.
- As is true for all queries, if you want a node-index value to be returned, then the client needs to explicitly specify it in the query. If a client does not request a node-index, then the server MUST NOT return it. The server MUST select data-nodes against the persisted index values, whether or not the query specifies the index. This behavior also applies to the index-column of systematically growing data-objects.

5.4 Mapping Growing Concepts to the Data Schema

The following table maps the growing data-object concepts defined in Sections5.2 and 5.3 to the v1.4.1 data schema.

Growing data-object Concept	MudLog (Randomly Growing)	Trajectory (Randomly Growing)	Log (Systematically Growing)
structural-range	startMd represents the minimum and endMd represents the maximum	mdMn represents the minimum and mdMx represents the maximum	If element direction has a value of: • "increasing" then startIndex or startDateTimeIndex represents the minimum and endIndex or endDateTimeIndex represents the maximum. • "decreasing" then startIndex or startDateTimeIndex represents the maximum and endIndex or endDateTimeIndex represents the minimum.
	When a depth or a date-and-time alternative is allowed, then only one consistent pair is allowed for any one query. That is, only a pair representing depth or a pair representing date-and-time may be specified. A client MUST NOT **[else error -458]** specify a combination of depth and date-time.		
node-index	In the geologyInterval sub-node, mdTop and mdBottom represent the minimum and maximum respectively of an index interval.	trajectoryStation/md represents the point index	The value in logData/data representing the indexCurve is the point-index.
data-node	In geologyInterval represents growing sub-nodes	trajectoryStation represents the growing sub-node	logData/data represents the growing sub-node.
column-definition	NA	NA	logCurveInfo represents the column-definition.
column-identifier	NA	NA	• For the column-definition the column-identifier is defined by logCurveInfo/mnemonic. • For the data-nodes the column-identifier is defined within logData/mnemonicList.
column-range	NA	NA	log/logCurveInfo minIndex or minDateTimeIndex represents the minimum index and maxIndex or maxDateTimeIndex represents the maximum index.

6. STORE Interface Functions

This chapter describes the STORE interface functions, which include those listed in Table 1. All WITSML servers must support the first two functions listed, which represent the basic requirements for a client and server to establish a connection and exchange data. All other functions are optional.

Table 1: STORE API Functions

STORE Function	Description/Purpose	For More Information
WMLS_GetVersion	Retrieves the version(s) of the data schema that are supported. **Required.** A server MUST support this function. A client must know the correct version of the data schema to use so it can correctly exchange data with the server.	Section 6.2, page 44
WMLS_GetCap	Get the server's Capabilities Object. **Required.** A server MUST support this function. A client must know the capabilities of a server to correctly interact with it.	Section 6.3, page 45
WMLS_AddToStore	Adds one new data-object to the server.	Section 6.4, page 47
WMLS_DeleteFromStore	Deletes one existing data-object (or subset of the data-object) from the server.	Section 6.5, page 50
WMLS_GetFromStore	Gets one or more data-objects from the server.	Section 6.6, page 54
WMLS_UpdateInStore	Updates one existing data-object on the server.	Section 6.7, page 89
WMLS_GetBaseMsg	Gets the fixed "base" description of a return value.	Section 6.8, page 96

6.1 Common Behavior of STORE Functions

This section lists some common behavior that you can expect across most or all functions in the STORE Interface.

6.1.1 Rights and Permissions

For all STORE interface functions that access or change data on a server (get, add, delete, update): in the server configuration for the site, the user MUST **[else error -414]** have rights to perform the requested operation on the data-object.

6.1.2 Important Restrictions on Processing Number and Types of Data-objects

The WMLS functions all specify a single WITSML data-object type (well, rig, etc) in their data-object type parameter. While the WMLS_GetFromStore function allows multiple occurrences of the same data-object type to be manipulated in one invocation, no functions allow the mixing of multiple data-object types. Thus, the XML documents transported by the STORE functions—both the data-objects and the Query templates—may contain multiple occurrences of a single data-object type, but not multiple data-object types.

6.1.3 Data-object Support and Mandatory Behaviors

- If a server claims to support a data-object for a specified function, then the server MUST support all elements defined in the data-object schema. The behavior of an operation within the API function is mandatory unless explicitly documented otherwise.
- For data-object selection support rules, see Chapter 7, page 97.
- A server is not required to populate optional elements.
- A server MUST populate mandatory elements.
- A client must accept all elements that a server sends, whether or not the client uses the data.

6.1.4 Use of UIDs across STORE Functions

Table 2 summarizes the requirements for use of UID attributes in the WITSML API functions (the requirements are also repeated in the section for each function). For more information on UIDs, see Section 2.2, page 15.

Table 2: UID Requirements for data-objects and API Functions

	GetFromStore	AddToStore	UpdateInStore	DeleteFromStore
object UID	optional	optional	required	required
parentage UID	optional	required	required	required
child UID	optional	required	required	optional

6.1.5 Case Sensitivity

WITSML servers preserve case in persisted string data items, but are not case sensitive when servicing STORE interface queries against those persisted data items. That is, string data values that differ only in character case are treated as being “equal”.

XML is NOT case sensitive.

For all functions exposed by the STORE interface, a server MUST preserve upper and lower case characters present in all data items that have an underlying data type of string.

For example, the country data item of the well data-object has a WITSML-defined data type that is derived from the W3C data type of string.

Therefore, if the country data item contains “Norway” when stored on a server, it will contain exactly the same case— “Norway” —when later retrieved by a client application.

However, when performing a query against a stored data item (including uids), a WITSML server MUST perform case-insensitive comparisons. If a client application wants to retrieve a well data-object whose name is equal to “My Best Well”, it could simply specify “my best well” and have returned to it data-objects containing:

```
      <name>My Best Well</name>
      <name>my best well</name>
      <name>MY BEST WELL</name>
      … or any other combination of character case.
```

Servers MUST perform uniqueness operations using a case-insensitive comparison.

6.1.6 Nullable Parameters

While all parameters have to be specified in each function, certain parameters MAY be specified with an empty value to indicate that a value was not given. This case generally applies to the OptionsIn and CapabilitiesIn parameters but is explicitly documented where applicable.

6.1.7 OptionsIn Parameter

An OptionsIn parameter provides a mechanism for a client to pass configuration information to the server; it is available for most functions. A server must support all OptionsIn parameters.

The OptionsIn string is patterned as a set of keyword=value pairs. Each function documents the keywords and values that are relevant to that function.

- A client MUST encode **[else error -411]** an OptionsIn parameter string using a subset of the encoding rules for HTML form content type application/x-www-form-urlencoded as specified at http://www.w3.org/TR/REC-html40/interact/forms.html#h-17.13.3.3. The encoding is constrained by allowing only semicolons for separators and by not allowing whitespace, which supports the following patterns:

```
   keyword1=value1
   keyword1=value1;keyword2=value2
   keyword1=value1;keyword2=value2;keyword3=value3
etc.
```

- A client MUST NOT **[else error -442]** specify an OptionsIn keyword that is not supported by the server (as defined in its Server Capabilities Object).
- A client MUST **[else error -440]** provide an OptionsIn keyword that is recognized by the function, and a client MUST **[else error -441]** specify a valid keyword value, as specified in this document. (For lists of keywords and valid values, see the sections for each of the STORE functions.)

6.1.8 Special Handling of Growing Data-objects

Except as noted in the individual sections for some functions in this chapter, standard rules apply for all other STORE functions for handling growing data-objects. For example, to add a data node in a randomly growing data-object, use WMLS_UpdateInStore and specify a node with a new uid value.

6.2 WMLS_GetVersion Function

Purpose: Returns a string containing the Data Schema Version(s) that a server supports.

A server MUST support this function.

When a client application wants to interact with a server, it first needs to determine which Data Schema Version(s) a server supports so the client can correctly exchange data with the server. To do this, the client uses the WMLS_GetVersion function.

After the client knows which Data Schema Versions are supported, the client can then pass a particular Data Schema Version using the OptionsIn parameter of WMLS_GetCap to request information about the server's capabilities.

This pseudo code example provides an overview of the function:

```
string = WMLS_GetVersion();
```

This function has no parameters.

6.2.1 Return Value

String. A comma-separated list of Data Schema Versions (without spaces) that the server supports. The server MUST list the supported Data Schema Versions, from oldest to newest, for example:

```
1.2.0.0,1.3.1.0,1.4.1.1
```

Each Data Schema Version number that the server returns MUST match the contents of the version attribute for the plural container element for each data-object in the schema. For more information on versioning, see Section 3.5, page 19.

6.3 WMLS_GetCap Function

Purpose: Returns the capServer object that describes the capabilities of the server for one Data Schema Version.

A server MUST support this function.

After the client uses WMLS_GetVersion to determine which Data Schema Versions that a server supports, the client can then use WMLS_GetCap to determine the functions (or capabilities) that the server supports.

The server returns the capServer object in the form of an XML document. To determine the capabilities of the server for all Data Schema Versions that it supports, the client must use multiple calls of WMLS_GetCap. For more information, see Section 4.2.4 Server Capabilities (capServer) Object, page 30.

This pseudo code example provides an overview of the function:

```
integer = WMLS_GetCap(
                    [in] string OptionsIn,
                    [out] string CapabilitiesOut,
                    out] string SuppMsgOut
                    );
```

6.3.1 Return Values

When a server finishes processing a request, it returns a numeric value. A positive value indicates a success; a negative value indicates an error. Throughout this section, return values are indicated for specific behaviors. For a complete list of values and their corresponding messages, see APPENDIX A: Defined Return Values, page 110.

6.3.2 Parameters (all required)

Parameter	Description/Purpose/Example
OptionsIn	• For information about formatting the OptionsIn parameter, see Section 6.1.7, page 43. • Input string. • Clients based on this STORE API specification MUST **[else error -424]** always pass the dataVersion OptionsIn parameter to the server, because the server may support multiple Data Schema Versions and the client needs to specify which Data Schema Version (s) it understands. • The keyword "dataVersion" with a value of a Data Schema Version specifies that capabilities information is desired for that particular Data Schema Version. The server MUST return a Capabilities Object that uses the namespace that is appropriate for the Data Schema Version. For example: dataVersion=1.4.1.1 requests information about Data Schema Version 1.4.1.1 using an API Schema Version 1.4.1 (the corresponding API Schema Version) Capabilities Object. To determine the Data Schema Version that a server supports, use the WMLS_GetVersion (see page 44). The client MUST **[else error -423]** specify a Data Schema Version that the server supports as defined by WMLS_GetVersion.
CapabilitiesOut	• Output string. • The server MUST send XML that conforms to the API capServer schema. For more information, see Section 4.2.4, page 30. • If the return code is positive, then a server returns its Server Capabilities object. If the return code is negative, then an error occurred. Note: The Server MUST return a non-null value.
SuppMsgOut	• Output string. Supplemental message text.

6.3.3 How it Works

Clients based on this STORE API specification MUST **[else error -424]** always pass the dataVersion OptionsIn parameter to the server, because the server may support multiple Data Schema Versions and the client needs to specify which Data Schema Version (s) it understands.

For example, WMLS_GetVersion may return “1.3.1.0,1.4.1.1” so a client that supports Data Schema Version 1.4.1.1 needs to specify “dataVersion=1.4.1.1”. The resulting Data Schema Version 1.4.1.1 CapabilitiesOut can contain something like this:

```
<capServers … version="1.4.1"> <!— API Schema Version ->
   <capServer apiVers="1.4.1"> <!— API Version ->
      <contact>
         <name>server contact name</name>
         <email>serverAdministrator@serverCompany.com</email>
         <phone>281-135-7924</phone>
      </contact>
      <description>server description</description>
      <name>John Smith</name>
      <vendor>INSITE server #1</vendor>
      <version>1.1.0.120</version> <!— Executable build ->
      <schemaVersion>1.4.1.1</schemaVersion> <!— Data Schema Version ->
      <changeDetectionPeriod>36</changeDetectionPeriod>
      <growingTimeoutPeriod dataObject="log">60</growingTimeoutPeriod>
      <growingTimeoutPeriod dataObject="mudLog">60</growingTimeoutPeriod>
      <cascadedDelete>false</cascadedDelete>
      <compressionMethod>gzip</compressionMethod>
      <function name="GetCap"/>
      <function name="GetFromStore">
         <dataObject>well</dataObject> <!— Supported Data-object ->
         <dataObject>wellbore</dataObject>
      </function>
   </capServer>
</capServers>
```

The server returns the API Version of "1.4.1" because that is the API Version that is associated with the requested “1.4.1.1” Data Schema Version. Each Data Schema Version is associated with only one API Version (i.e., the active API at the time of the schema release). This assumption is made because the current interface requires that a Data Schema Version be used with only one API Version (including any corrigendum). Correspondingly, the API Version requires a specific API Schema Version. The API Schema Version associated with this API Version is "1.4.1". The Data Schema Versions associated with this API Version are “1.4.1.0” and any subsequent data schemas until the next API Version.

For more information about the various WITSML component versions and the relationship among them, see Section 3.5, page 19.

6.4 WMLS_AddToStore Function

Purpose: Adds one WITSML data-object to the server.

This section describes the function's operational parameters and general behaviors for how it works.

This pseudo code example provides an overview of the function:

```
integer = WMLS_AddToStore(
                                    [in] string WMLtypeIn,
                                    [in] string XMLin,
                                    [in] string OptionsIn,
                                    [in] string CapabilitiesIn,
                                    [out] string SuppMsgOut
                                    );
```

6.4.1 Return Values

When a server finishes processing a request, it returns a numeric value. A positive value indicates a success, and a negative value indicates an error. Throughout this section, return values are indicated for specific behaviors. For a complete list of values and their corresponding messages, see APPENDIX A: Defined Return Values, page 110.

6.4.2 Parameters (all required)

<table>
<tr><th>Parameter</th><th>Description/Purpose/Example</th></tr>
<tr><td>WMLtypeIn</td><td>

Input string. One WITSML data-object type (see the specific WITSML data schema for the objectType).
A client MUST [else error -407] specify a non-empty value.
The objectType specified in this parameter MUST [else error --486] match the objectType in the XMLin.
The objectType MUST [else error -487] match an objectType supported by the server (as defined in the capServer).

</td></tr>
<tr><td>XMLin</td><td>

Input string. The WITSML data-object to be added.
A client MUST [else error -408] specify a non-empty value.
The client MUST [else error -409] pass XML that conforms to the derived "write" schema (no compression specified) where child uids and parentage-pointers are mandatory or a client MUST [else error -426] validate against the derived "write" schema after uncompressing.

</td></tr>
<tr><td>OptionsIn</td><td>

For information about formatting the OptionsIn parameter, see Section 6.1.7, page 43.
Input string
If the client sends an empty string; the server MUST use default values of all options.
compressionMethod specifies a type of compression used by the client in the request. (The server MUST support HTTP compression in the response.)
A server MAY support client compression. If a server supports client compression, then the server MUST specify compressionMethod in its server Capabilities Object. See Section 4.2.4, page 30.
After a client applies a compression method, the client must [else error -479] encode the resultant byte array into the XMLin/QueryIn string using base64Binary.

Currently recognized compression methods include:
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>None (DEFAULT)</td><td>Specifies that client is not applying compression to QueryIn.</td></tr>
<tr><td>gzip</td><td>Specifies that the client is using gzip compression.</td></tr>
</table>
</td></tr>
</table>

Parameter	Description/Purpose/Example
CapabilitiesIn	• Input string. The client's Capabilities Object (capClient) to be sent to the server. For more information, see section 4.2.2, page 28. • A server MUST interpret an empty string to indicate that no value was given. • The client MUST **[else error -466]** send XML that conforms to the API capClient schema (for an example, see Section 4.2.3.2, page 30).
SuppMsgOut	• Output string. Supplemental message text. • If the server creates a uid for the data-object, then the server MUST return that created uid value in this string. If there are any errors, the Server MUST NOT return the uid value. • The server may add extra information after the UID and if it does, it must put a space (delimiter) following the UID. If the UID is included, it must be first in the string.

6.4.3 UID Requirements

To identify, retrieve and update WITSML data-objects, each data-object has a machine-readable unique identifier (UID). For more information on UIDs, see Section 2.2, page 15. This table lists requirements for using UIDs within this function.

UID Requirements for AddToStore	
Object	**Optional/Required?**
Object UID	optional
Parentage UID	required
Child UID	required

6.4.4 How it Works

- A client adds a data-object to the server by sending the data-object type (in WMLtypeIn) and the data-object's unique identifier (in XMLin); the client does not pass unique identifiers as separate parameters. The client can add both header rows and log data rows at the same time. **Example:** The following XMLin adds a new well data-object with the UID of "new1":

```
<wells xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
       <well uid="new1">
              …
       </well>
</wells>
```

- In the XMLin file, the client MUST **[else error -401]** include the plural container data-object as its root XML element, even though it encloses only one WITSML data-object. **Example:**

```
<wells..> [mandated even though there is only one <well> element]
       <well uid="new4">
              …
       </well>
</wells>
```

- A client MUST NOT **[else error -405]** specify a data-object with the same type and unique identifier that already exists in the persistent store (if updating an existing data-object, use WMLS_UpdateInStore function).
- If the client does not define the uid value for the data-object in the XMLin file, the server MUST create one and (if no errors occur) return the created uid value to the client in SuppMsgOut. For more information about uids, see Section 2.2.1, page 15.

- In the XML file, the client MUST **[else error -406]** define all other parentage-pointers and lower level (child) uid values. The client MUST **[else error -464]** define each lower level child uid value to be unique within the context of its nearest recurring parent node.
- The client MUST **[else error -481]** specify parent identifiers of objects that actually exist in the server.
- The client MUST **[else error -413]** add only data-object types that the server supports, as defined in its Capabilities Object.
- The client MUST **[else error -412]** provide a data-object in XMLin that is compliant to the write schema.

6.4.4.1 Case

The server MUST store upper/lowercase string data values present in the XMLin string exactly as supplied, “preserving their case.” For subsequent invocations of WMLS_GetFromStore, the server MUST return the string data values in their originally supplied case.

- When a client is adding an object to a server, the client MUST **[else error -478]** ensure the case of the Parent UID(s) match the case of the UID(s) of the object already in the server.

For more information on case sensitivity in functions, see Section 6.1.5, page 42.

6.4.4.2 Units of Measure (UOM)

For more information about units of measure, see Chapter 8, page 99.

- The client MUST **[else error -453]** always specify the uom for all measure data.
- The client MUST **[else error -443]** specify a value of uom that matches an annotation attribute from the WITSML Units Dictionary XML file.

6.4.4.3 Nullable Parameters

If a client sends a value of NaN in a numeric field to the server, then the server MUST convert it to null or “not given”.

6.4.4.4 Additional Information for Growing Data-objects

For definitions of growing data-objects, see Chapter 5, page 36.

If a server constrains the amount of data that can be added in growing data-objects, then it MUST define those limits in the server Capabilities Object by setting attributes maxDataNodes and maxDataPoints as appropriate for each data-object. For more information about server Capabilities Object, see Section 4.2.4, page 30.

The value of maxDataNodes represents the maximum number of data-nodes that a function can handle for the specified data-object:

- For a systematically growing data-object, it represents the number of data rows.
- For a randomly growing data-object, it represents the total number of sub-objects, regardless of type.

The client MUST NOT **[else error -456]** attempt to add more data than is allowed by maxDataNodes and maxDataPoints values.

For systematically growing data-objects, the data MUST NOT **[else error -463]** contain multiple nodes with the same index.

6.4.5 Best Practices

- Clients should always use pointers (e.g., uidWell, uidRef) whose value exactly matches the related uid value.

6.5 WMLS_DeleteFromStore

Purpose: Permanently deletes one WITSML data-object from the data store.

The client specifies the data-object to be deleted in the WMLtypeIn and QueryIn parameters. A data-object with the same type and unique identifier(s) MUST **[else error -433]** already exist in the persistent store.

This pseudo code example provides an overview of the function:

```
integer = WMLS_DeleteFromStore(
                                    [in] string WMLtypeIn,
                                    [in] string QueryIn,
                                    [in] string OptionsIn,
                                    [in] string CapabilitiesIn,
                                    [out] string SuppMsgOut
                                    );
```

6.5.1 Return Values

When a server finishes processing a request, it returns a numeric value. A positive value indicates a success, and a negative value indicates an error. Throughout this section, return values are indicated for specific behaviors. For a complete list of values and their corresponding messages, see APPENDIX A: Defined Return Values, page 110.

6.5.2 Parameters (all required)

Parameter	Description/Purpose/Example
WMLtypeIn	• Input string. One WITSML data-object type (see the specific WITSML data schema). • A client MUST **[else error -407]** specify a non-empty value.
QueryIn	• Input string. The WITSML data-object to be deleted. • A client MUST **[else error -408]** specify a non-empty value. • Because QueryIn can contain empty elements, it cannot be schema validated but otherwise the client MUST **[else error -409]** ensure that the XML conforms to a derived “delete” schema where all elements and attributes are optional, except for all data-object uids and parentage-pointers, which are mandatory.

OptionsIn	• For information about formatting the OptionsIn parameter, see Section 6.1.7, page 43.
	• Input string
	• If the client sends an empty string; the server MUST use the default values of all options.
	• The keyword of ‘cascadedDelete’ specifies whether or not the server enforces cascaded deletions.
	  o If the client specifies a value of “true”, when a server deletes a data-object, the server MUST delete all data-objects whose identity depends on that data-object. The user MUST **[else error -414]** have rights to delete the dependent data-objects. For example, deleting a well data-object would result in the deletion of all dependent wellbore data-objects and any data-objects that depend on those wellbore data-objects. Deletion means that the dependent data-object cannot be "found" by the WITSML API interface, even though some data may continue to exist in a back-end database. Non-parentage pointers (e.g., uidRef) are a data quality concern such that servers MUST ignore their values.
	  o If a client specifies a value of “false”, then a server MUST NOT delete a data-object until all dependent data-objects have been deleted. This is the default value.

Parameter	Description/Purpose/Example
	o A server MAY support this option. If the server supports this option, then the server MUST specify cascadedDelete as “true” in its server Capabilities Object. For more information on deleting data-objects and sub-nodes, see Section 6.5.4, page 51.
CapabilitiesIn	• Input string. The client’s Capabilities Object (capClient) to be sent to the server. • A client MUST **[else error -472]** identify itself to the server using the HTTP user-agent field. The client must pass its product name (preferably the short name) and version number as defined in the W3C standard: http://www.w3.org/Protocols/HTTP/HTRQ_Headers.html#user-agent. • The server MUST interpret an empty string to indicate that no value was given. • The Client MUST **[else error -466]** ensure that the XML conforms to the API capClient schema.
SuppMsgOut	• Output string. Supplemental message text.

6.5.3 UID Requirements

To identify, retrieve and update WITSML data-objects, each data-object has a machine-readable unique identifier (UID). For more information on UIDs, see Section 2.2, page 15. This table lists requirements for using UIDs within this function.

UID Requirements for DeleteFromStore	
Object	**Optional/Required?**
Object UID	required
Parentage UID	required
Child UID	optional

6.5.4 How it Works

- The client specifies the data-object to be deleted using a query template: specifically the WMLtypeIn (object type) and QueryIn (the actual data-object) parameters. The client MUST **[else error -415]** specify the uid for the data-object and its parents. For example, the following code deletes a specific wellbore data-object as identified by the uidWell and uid attributes:

```
<wellbores xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
      <wellbore uidWell="12345" uid="01" />
</wellbores>
```

- A server MUST use a caseless compare against uid values (as it does for all other values).
- The server MUST support the data-object as defined in its Capabilities Object.
- If a client does not invoke cascading deletes, then the client MUST **[else error -432]** only request deletion of bottom-level data-objects such that all child data-objects are deleted before the parent is deleted. For example, it is necessary to delete a child wellbore before deleting the parent well.
- To eliminate the likelihood of accidental deletion of multiple data-objects, the client MUST **[else error -415]** specify in the query template the UID of the one data-object to be deleted. The client MUST NOT **[else error -444]** specify more than one data-object to be deleted in a query.

  Example: You cannot request WMLS_DeleteFromStore—in one step—to delete all the logs where <indexType> is “measured depth”:

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
      <log>
            <indexType>measured depth</indexType>
      </log>
</logs>
```

This query will fail because no uid is specified.

- If the client application needs to delete multiple data-objects based on selection criteria other than the UID, it can first use WMLS_GetFromStore to retrieve the UIDs of the data-objects based on the necessary criteria, and then repetitively pass each returned data-object containing the UIDs to WMLS_DeleteFromStore.

6.5.4.1 How to Delete Sub-nodes

For more information about growing data-objects, see Chapter 5, page 36.

To delete a sub-node, a client must specify an empty sub-node; the server MUST delete it. This standard behavior also applies to randomly growing data-objects (which, by definition, have a uid).

For the purpose of the following discussion, a container-element is an element that contains other elements (as opposed to a value). A non-container-element is an element that has a value and/or possibly an attribute but not other elements.

- For deletion of sub-nodes—as as opposed to the whole data-object—a server SHOULD only require that a user have “update” rights, as opposed to “deletion” rights.
- To delete an attribute, the client specifies an empty attribute and the server MUST delete that attribute. **Exceptions to this rule, the client:**
  - MUST NOT **[else error -416]** specify an empty uid attribute; that is, you cannot delete a uid.
  - SHOULD NOT specify an empty uidRef attribute, unless the related item has been deleted.
  - MUST NOT **[else error -417]** specify an empty unit of measure (uom) attribute.
- For an empty non-container element with no unique identifier in the schema, a server MUST delete the values and any attributes. **Example:** this code deletes the country element in the specified well data-object:

```
<wells xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
      <well uid="12345">
            <country />
      </well>
</wells>
```

- For an otherwise empty recurring element with a uid value, the server MUST delete that specific occurrence. **Example:** this code deletes a single trajectoryStation in the specified data-object:

```
<trajectorys xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
      <trajectory uidWell="W-12" uidWellbore="B-01"
                  uid="pe84e">
            <trajectoryStation uid="123" />
      </trajectory>
</trajectorys>
```

- A client MUST NOT **[else error -418]** specify an element (recurring or not) with a uid in the schema without specifying a uid value. For example, this code is invalid because a uid value is not specified:

```
<trajectorys xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
      <trajectory uidWell="W-12" uidWellbore="B-01"
                  uid="pe84e">
            <trajectoryStation/>
            <trajectoryStation uid=""/>
      </trajectory>
</trajectorys>
```

Invalid because the uid values are not specified for trajectoryStation.

- For an empty recurring element with no uid in the schema, a server MUST delete all occurrences. EXCEPTION: In systematically growing data-objects (e.g., log), the client MUST NOT **[else error -419]** specify an empty value for logData. The recurring nature of logData is only a sparse view of the actual data. For example, this code deletes all nameFormation values in the specified geologyInterval:

```
<mudLogs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
      <mudLog uidWell="W-12" uidWellbore="B-01" uid="pe84e">
            <geologyInterval uid="12399">
                  <nameFormation/>
            </geologyInterval>
      </mudLog>
</mudLogs>
```

- A client MUST NOT **[else error -419]** specify an empty non-recurring container-element with no unique identifier in the schema. Instead, the client should delete each of the items within the container element.
- A client MUST NOT **[else error -420]** specify an empty node for a non-recurring element or attribute that is mandatory in the write schema.
- The client MAY delete a recurring element that is mandatory in the write schema, but a client MUST **[else error -421]** retain at least one node after the deletion.
- The client MUST **[else error -418]** specify the unique identifier of all parent nodes, but only the leaf node (and its children) will be deleted. The server MUST retain the parent nodes.

Additional Information for Deleting Sub-nodes for Systematically Growing Data-objects

- If a client specifies a structural-range value:
  - The server MUST delete all data-nodes with a node-index that is greater than or equal to the minimum range value and less than or equal to the maximum range value.
  - If a client specifies only the minimum range value, then the server MUST delete all data-nodes with a node-index value that is greater than or equal to the range value.
  - If a client specifies only the maximum range value, then the server MUST delete all data-nodes with a data index value that is less than or equal to the range value.
- If a client specifies a column-identifier, then the server MUST delete all information associated with that column.
  - If a client specifies a structural-range with this option, then server MUST only delete the specified subset of the column. For this function, the client specifies the column-identifier in the header column description, as opposed to a data-column-list.
  - The client MUST NOT **[else error -437]** specify the data-column-list.

6.6 WMLS_GetFromStore Function

Purpose: Returns one or more WITSML data-objects from the server. The server returns data-objects in an XML document containing a root plural data-object element (e.g., <wells>) enclosing one or more singular data-object elements (e.g., <well>). The client identifies the data-objects to be retrieved using a query template that is passed to the function.

This pseudo code example provides an overview of the function:

```
integer = WMLS_GetFromStore(
                                    [in] string WMLtypeIn,
                                    [in] string QueryIn,
                                    [in] string OptionsIn,
                                    [in] string CapabilitiesIn,
                                    [out] string XMLout,
                                    [out] string SuppMsgOut
                                    );
```

6.6.1 Return Values

When a server finishes processing a request, it MUST return a numeric value (return value). To indicate a successful operation, the server MUST return a positive value. To indicate an error, a server MUST return a negative value (and, in this case, the server returns no data). Throughout this section, return values are indicated for specific behaviors. For a complete list of values and their corresponding messages, see APPENDIX A: Defined Return Values, page 110.

If a server truncates the return of growing sub-object nodes for any non-error reason, the server MUST return a value of +2 to indicate a successful operation and to indicate that more data is available. This behavior also applies when multiple growing data-objects are returned and one or more of them are truncated.

6.6.2 Parameters (all required)

Parameter	Description/Purpose/Example
WMLtypeIn	• Input string. One WITSML data-object type (see the specific WITSML data schema). • A client MUST **[else error -407]** specify a non-empty value.
QueryIn	• Input string. A query template that specifies the data-object(s) to be returned. For more information about templates, see Section 6.6.5, page 65. • The client MUST **[else error -408]** specify a non-null value. • If the QueryIn contains empty elements, then server cannot schema validate it; otherwise the client MUST **[else error -410]** ensure that the XML conforms to a derived “read” schema where all elements and attributes are optional. The server may choose not to test for this condition. • If the OptionsIn keyword of ‘returnElements’ is used with a value other than “requested”, then the contents of QueryIn must do one of the following: o Client MUST **[else error -409]** validate against the derived “read” schema (no compression specified) OR o Client MUST **[else error -426]** validate against the derived “read” schema after uncompressing.
OptionsIn	• For information about formatting the OptionsIn parameter, see Section 6.1.7, page 43. • Input string. • If the client sends an empty string, then the server MUST use default values of all options. • For the list of OptionsIn valid values, see Section 6.6.2.1, page 55.

Parameter	Description/Purpose/Example
CapabilitiesIn	• Input string. The client's Capabilities Object (capClient) to be sent to the server. For more information, see section 4.2.2, page 28. • The server MUST interpret an empty string to mean that no value was given. • The client XML MUST **[else error -466/optional]** conform to the API capClient schema. The server may choose not to test for this condition.
XMLout	• Output string. The returned WITSML data-object(s) is an XML document. • If the return code is positive, then the server MUST return a non-null value. • The server MUST return an XML schema that conforms to a derived read schema where all choices, elements and attributes are optional.
SuppMsgOut	• Output string. Supplemental message text.

6.6.2.1 OptionsIn Keywords

This section defines the keyword values and their use for the OptionsIn parameter.

Keyword	Description/Purpose and Valid Values
returnElements	• Indicates which elements and attributes are requested to be returned in addition to data-object selection items. • This keyword works in conjunction with WMLS_GetFromStore standard query templates. For more information, see Section 6.6.5, page 65. • All values except "requested" are an alternative to specifying empty values for data item selection such that the server MUST treat the template as if the indicated elements and attributes had explicitly been specified in the template. • A server MUST support this option. Valid values for this keyword:

Value	Returns/Description
Requested (DEFAULT)	Only explicitly specified data-object-selection and data item selection items.
all	• May be used for all data-object types. • Returns all elements and attributes. • For growing data-objects, only one data-object can be selected per query but multiple individual queries may be included inside the query plural data-object.
id-only	• May be used for all data-object types. Returns: • Parentage uids and names (if any): o uidWell and nameWell (if it exists in schema). o uidWellbore and nameWellbore (if it exists in schema). • data-object uid and name.
header-only	• Used only for growing data-objects. Query MUST NOT **[else error -425]** specify this option for a non-growing data-object. Returns: • Non-growing elements and attributes of a data-object (i.e., growing data will be excluded).
data-only	• Used only for growing data-objects. Query MUST NOT **[else error -425]** specify this option for a non-growing data-object. • Query MUST **[else error -475]** specify a subset of one

<table>
<tr><th>Keyword</th><th>Description/Purpose and Valid Values</th></tr>
<tr><td></td><td>
<table>
<tr><td></td><td>growing data-object per query, but multiple individual queries may be included inside the query plural data-object (see examples in 0, page 79.).
Returns:
• Only the growing elements and any required context (e.g., mnemonicList, unitList, etc.).</td></tr>
<tr><td>station-location-only</td><td>• A specialization of “data-only” for used only for the trajectory data-object. Query MUST NOT [else error -425] specify this value for any data-object other than a trajectory.
Returns:
• Any data-object selection-criteria (e.g., uid) plus all trajectoryStation elements which meet any range-selection criteria.
• Only the following trajectoryStation elements: dTimStn, typeTrajStation, md, tvd, incl, azi, location.</td></tr>
<tr><td>latest-change-only</td><td>• Used for changeLog data-object only. A query MUST NOT [else error -476] specify this value for any data-object other than the changeLog. For more information on the changeLog data-object, see Chapter 9, page 101. For changeLog data-object standard query templates, see Section 6.6.5.17, page 84.)
• Returns everything except for changeHistory elements.</td></tr>
</table>
</td></tr>
<tr><td>maxReturnNodes</td><td>
• Specifies that a server MUST constrain the number of returned growing sub-nodes in each query to a count that is less than or equal to maxReturnNodes. The query MUST [else error -402] specify a whole number greater than zero.
• DEFAULT BEHAVIOR: When this keyword is not specified, the default behavior is to return everything up to the server limits.
• A server that declares support for a growing data-object MUST support this option.
<table>
<tr><th>WITSML data-object</th><th>What maxReturnNodes is applied to</th></tr>
<tr><td>trajectory</td><td>count of trajectoryStation nodes</td></tr>
<tr><td>mudLog</td><td>total count of parameter and geologyInterval nodes</td></tr>
<tr><td>log</td><td>count of logData/data nodes</td></tr>
</table>
• If the value times the number of queries is greater than the number of nodes indicated by the server capability's maxDataNodes or maxDataPoints (which apply to the whole request), then the server uses the lower of this value or its internal constraints <reference to algorithms how these are computer> (Note that the value of maxDataPoints does not apply to randomly growing data-objects.)
• For randomly growing data-objects, the server MUST ensure that the truncated nodes have an index (or start index for intervals) that is greater than the maximum (start) index of the returned data.
</td></tr>
<tr><td>intervalRangeInclusion</td><td>
• Specifies the criteria for inclusion of a growing data-object node representing an index interval (e.g., geologyInterval) when a structural-range is specified. For more information, see Section 6.6.4.3, page 61.
• A server that declares support for an interval-based growing data-object (e.g., mudLog is currently the only one) MUST support this option.
• Does not apply to point-based growing data-objects, such as log and trajectory, only mudLog. (For examples, see Section 14.2.3, page 124.)
</td></tr>
</table>

<table>
<tr><th>Keyword</th><th>Description/Purpose and Valid Values</th></tr>
<tr><td></td><td>Valid values for this keyword:
<table>
<tr><th>Value</th><th>Requests that the node be included if:</th></tr>
<tr><td>minimum-point
(DEFAULT)</td><td>The interval minimum is within the range.</td></tr>
<tr><td>whole-interval</td><td>If the whole interval is within the range.</td></tr>
<tr><td>any-part</td><td>Any part of the interval overlaps the range.</td></tr>
</table></td></tr>
<tr><td>compressionMethod</td><td>

Specifies a type of compression used by the client in the request. (The server MUST support HTTP compression in the response.)
A server MAY support client compression. If a server supports client compression, then the server MUST specify compressionMethod in its server Capabilities Object. See Section 4.2.4, page 30.
After a client applies a compression method, the client must [else error -479] encode the resultant byte array into the XMLin/QueryIn string using base64Binary.

Currently recognized compression methods include:
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>none
(DEFAULT)</td><td>Specifies that client is not applying compression to QueryIn.</td></tr>
<tr><td>gzip</td><td>Specifies that the client is using gzip compression.</td></tr>
</table></td></tr>
<tr><td>requestObjectSelectionCapability</td><td>

When a client specifies this option with a value other than “none”:

It is requesting that the server return in XMLout a template that defines the items supported by the server for object selection.
A client MUST NOT [else error -427] specify another option and the the QueryIn parameter MUST [else error -428] be the Minimum Query Template (see Section 4.1.6, page 27 or the example below). The server MUST return XMLout that conforms to the derived read schema, but the returned values will be meaningless.

<pre>
<wells xmlns="http://www.witsml.org/schemas/1series"
version="1.4.1.1">
      <well/>
</wells>
</pre>

A server MUST support this option.
For more details on the server requirements for this option, see Chapter 7, page 97.

Valid values for this keyword:
<table>
<tr><th>Value</th><th>Behavior</th></tr>
<tr><td>none
(DEFAULT)</td><td>Indicates normal behavior.</td></tr>
<tr><td>true</td><td>Requests that the returned template define all items the server supports for data-object selection.</td></tr>
</table></td></tr>
<tr><td>requestLatestValues</td><td>

DEFAULT: normal log behavior. A server MUST support this option.
If specified, return the latest n values from each curve in a log data-object.
VALID VALUE: A whole number from 1 to the maximum value specified in a server’s Capabilities Object in the maxRequestLatestValues element (see Section 4.2.4, page 30).
If the query specifies requestLatestValues, then the server ignores any range in start and end index.
See example below.
</td></tr>
</table>

Keyword	Description/Purpose and Valid Values
	NOTE: This option applies only to the log data-object. It is ignored by all other data-objects.
requestPrivateGroupOnly	• A server MUST support this option. • Boolean. DEFAULT = false. • Used to query objects that have the field *privateGroupOnly* set in the common data. • If false or not specified, standard queries will not return these objects. • If true, standard queries will return only these objects.

Example: requestLatestValues

The requestLatestValues is used to replace the realtime data-object (which was a snapshot of a log). If requestLatestValues=1, it replaces the realtime object, showing the last (latest) curve added to a log. Additionally you can specify larger numbers (up to the server's maxRequestLatestValues) to see several of the most recent curves added.

Overview:

- 1 log with 3 curves and 1 index (mdepth)
- Curve 1, "GR", has a range from 0 to 1000
- Curve 2, "TQ on btm", has a range from 0 to 2000
- Curve 3, "TQ off btm", has a range from 0 to 3000
- The data has a stepIncrement of 0.1, so you can infer the expected indexes.

Two logData sections are included in the example results:

- One shows the result of the client issuing a requestLastValues=5
- The other shows the result of the client issuing a requestLastValues=1

```
<!-- result if OptionsIn requestLatestValues=5

             Null values in a <data> row indicate that this is not a last ( index
and value ) for this Curve.

             This example has the first 5 rows have non-null values for the 'GR' (
indicating that these rows are only describing the last values for 'GR' ) the second
5 rows have non-null values for 'TQ on btm' and the last 5 rows have a non-null
value of 'TQ off btm'

        -->
        <logData>
            <mnemonicList>Mdepth,GR,TQ on btm,TQ off btm</mnemonicList>
            <unitList>m,gapi,kft.lbf,kft.lbf</unitList>
            <data> 999.6, 1.00,,</data>
            <data> 999.7, 1.01,,</data>
            <data> 999.8, 1.02,,</data>
            <data> 999.9, 1.03,,</data>
            <data>1000.0, 1.04,,</data>
            <data>1999.6,, 20.00,</data>
            <data>1999.7,, 20.01,</data>
            <data>1999.8,, 20.02,</data>
            <data>1999.9,, 20.03,</data>
            <data>2000.0,, 20.04,</data>
            <data>2999.6,,, 300.00</data>
            <data>2999.7,,, 300.01</data>
```

```
            <data>2999.8,,, 300.02</data>
            <data>2999.9,,, 300.03</data>
            <data>3000.0,,, 300.04</data>
        </logData>
```

```
        <!-- result if OptionsIn requestLatestValues = 1 -->
        <logData>
            <mnemonicList>Mdepth,GR,TQ on btm,TQ off btm</mnemonicList>
            <unitList>m,gapi,kft.lbf,kft.lbf</unitList>
            <data>1000.0, 1.04,,</data>
            <data>2000.0,, 20.04,</data>
            <data>3000.0,,, 300.04</data>
        </logData>
```

6.6.3 UID Requirements

To identify, retrieve and update WITSML data-objects, each data-object has a machine-readable unique identifier (UID). For more information on UIDs, see Section 2.2, page 15. This table lists requirements for using UIDs within this function.

UID Requirements for GetFromStore	
Object	**Optional/Required?**
Object UID	optional
Parentage UID	optional
Child UID	optional

6.6.4 How it Works

- The server MUST support the data-object as defined in its Capabilities Object.
- Output parameters are valid only if Return Value is greater than zero. If Return Value is less than zero, the XMLout is not valid and only SuppMsgOut is valid.
- The server performs no locking of the data-object.
- A server MUST use a caseless compare against uid values (as it does for all other values).
- The QueryIn template will not be schema compliant if it contains empty values. If it does not contain empty values, the client MUST **[else error -409]** ensure that the QueryIn template be schema compliant against a derived “read” schema where all elements and attributes are optional. To effectively construct a query with empty values, use the OptionsIn keyword returnElements with a value other than “requested.”
- If all of the criteria are not satisfied in an individual (singular) query, then the server MUST return no data-object for that query, but must return a document that contains only the plural container element.
- If multiple queries are contained in the template and one query fails, then all queries fail and the server MUST return the error code of the query that failed.
- A client MUST accept all elements that a server sends, whether or not the client uses the data.
- For the commonData and commonTime elements dTimCreation and dTimLastChange, the server MUST treat a request for comparisons as "greater than" NOT "equal". This behavior is an exception to the standard STORE behavior of "equal" comparisons.
- For a full query template result (for example, OptionsIn keyword returnElements = all), the server MUST return results that are compliant to the write schema.

6.6.4.1 Information about Query Results

- WITSML servers can return query results in any order. When multiple queries are present in a query template, the server MAY return the data-objects in any order.
- The server MUST return the elements in the order defined by the schema.
- The ability to submit multiple queries for the same data-object type is provided for efficiency; there is otherwise no relationship between the queries. Each is a separate, standalone query that will result in zero or more well data-objects being selected for each, which may result in the same data-object being returned for multiple queries.
- For data-object(s) selected by multiple queries in one call, the server MUST return separate data-objects in one plural container. The returned data-objects will not be explicitly identified with the

query that requested them. It is the responsibility of the client to determine which data-object(s) were returned as a result of which query, if such a correlation is necessary.

- If no value is available for an element or attribute, then the server MUST return nothing. That is, the server MUST NOT return empty values.
- The order of the recurring data within a data-object is server dependent. The order in which a client added components will not necessarily be retained, unless the schema specifies some significance of the order. Unless the data-object explicitly specifies that it has a particular order (e.g., log/direction), a client SHOULD NOT assume any particular order in the results. If a client wants nodes in a particular order, then it must sort the nodes itself.
- The server MUST return a document that contains a plural container data-object as its root XML element, enclosing zero or more singular data-object elements:

```
<wells xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
       <well…>
              …
       </well>
       <well…>
              …
       </well>
</wells>
```

- If the query template results in no data-objects being returned, then the server MUST return a document that contains only the plural container element:

```
<wells xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1" />
```

- A server MUST NOT return a value of NaN. If a client sends a value of NaN to the server, then the server MUST convert it to null or “not given”, depending on the context.
- The normal behavior for a server is to return only “what you ask for.” That is, for an element to be returned, it must exist in the query template. This includes uom attributes (see Chapter 8 Units of Measure, page 99).
  - For the special case of <customData> (see Chapter 10 Custom Data, page 109), if the client specified only the customData element, then the server MUST return all sub-nodes. Querying sub-nodes of customData is server dependent.
  - For the special category of growing data-objects and their special behavior, see Section 6.6.4.3, page 61 and Chapter 5, page 36.

6.6.4.2 Units of Measure (UOM)

For information on handling units of measure, see Chapter 8, page 99.

6.6.4.3 Special Handling for Growing Data-objects

For definitions of growing data-objects and related schema elements, and explanation of why growing data-objects require special handling, see Chapter 5, page 36.

How Clients Must Specify Data Ranges and How Servers Must Respond

Determining Selected-nodes

The selected-nodes are those persistent data-nodes that are candidates to be returned. However, no one piece of information defines what data-nodes are actually selected; it is a combination as described in this section.

- If a client does not specify a structural-range value, then the server MUST include all data-nodes as selected-nodes.
- If a client specifies a structural-range value, then the server MUST interpret it to include as selected-nodes all data-nodes where the node-index values are at or within the index range. If a

client does not specify a minimum range value, then the server MUST assume a value of negative infinity. If a client does not specify a maximum range value, then the server MUST assume a value of positive infinity.

- For a point index (see schema), the server MUST include as selected-nodes those data nodes where the node-index is greater than or equal to the minimum range value AND less than or equal to the maximum range value.
- The server MUST include as selected-nodes those intervals that meet the conditions specified in below (For an example, see Section 14.2.3.1, page 125).

If the OptionsIn keyword intervalRangeInclusion is this value:	And these conditions are true:
whole-interval	• The minimum node-index is greater than or equal to the minimum range value, and • The maximum node-index is less than or equal to the maximum range value. That is, the whole interval is at or within the range.
minimum-point (DEFAULT VALUE)	• The minimum node-index is greater than or equal to the minimum range value and less than or equal to the maximum range value. That is, the server will ignore the maximum node-index and the server will treat the interval as a point for this test.
any-part	• The maximum node-index is greater than or equal to the minimum range value, and • The minimum node-index is less than or equal to the maximum range value. That is, if any part of the interval overlaps the range.

- The server MUST return only selected-nodes that match the structural-range values.
- The server MUST return requested header data that represents the range of selected-nodes in the results.
- If there are no selected-nodes, then the server MUST return nothing for that query because not all criteria will have been met.

For Randomly Growing Data-objects

- If a client specifies a value for a node-index element AND a value for structural-range value, then the server MUST ignore the node-index value. The server MUST interpret the node-index element to be a standard STORE query request for a value of that element.
- For a randomly growing data-object (i.e., mudLog startMd/mdTop, trajectory mdMn/mdMx), if the client specifies structural-range client values, then the server's returned values represent the minimum and maximum values within that range, whether returned (i.e., data requested) or in-store (i.e., only header requested). "Minimum" means the nearest to the surface measured along the borehole, which is in contrast to the returned values in a systematically growing data-object (i.e., log) where the values represent the first and last values. The difference is because systematically growing data-objects are ordered while randomly growing data-objects are not necessarily sorted (i.e., the actual order is server dependent).

For Systematically Growing Data Objects

Reminder: Systematically growing data-objects have no UIDS; they are essentially tables. As such, column-identifier and column-range elements are used to specify a structural-range and selection criteria.

- The structural-range and column-identifiers represent cellular-selection criteria (i.e., which cells from a table) and the server MUST AND them together. The server MUST also AND any other cellular selection criteria (e.g., a specific data value or header value). The server MUST AND any cellular-selection criteria with any data-object-selection criteria.
- If a the client specifies a column-range element:
  - Then the server MUST ignore it. The server MUST interpret the element as a standard STORE query request for a value of that element. The server MUST return a value that reflects the selected-nodes.
  - If there are no selected-nodes for an individual table, then the server MUST return nothing for that table.
- If a client specifies an empty column-identifier value in the column-description, then the server MUST interpret it as a request for header information about all columns. Note that a returnElements parameter value, such as *all*, implies an empty column-identifier if the client specifies none.
- If a client specifies an empty column-identifier in the data-column-list, then the server MUST interpret it as a request for data values for all columns.
- If both data and header information are desired, then a client MUST **[else error -477]** specify both the column-description and data sections. A client MUST **[else error -460]** also specify the same column-identifiers in both sections.
- If a client specifies a column-definition section, then a client MUST **[else error -461]** also specify a mnemonic element. If a client specifies a data-node section, then a client MUST **[else error -462]** also specify the mnemonicList element.
- When the client requests data-nodes:
  - The server MUST return data rows for all selected-nodes that contain non-null data.
  - If no non-null data exists for the requested columns in the selected-nodes, then the server MUST return nothing for that query. The index value is not considered to be a data value.
  - If a column only contains null data within the selected-nodes, then the server MUST return nothing (neither header nor data) for that column. This also means that the column-identifier MUST NOT exist in the data-column-list.
- In a systematically growing data-object (e.g., log), the client MUST **[else error -429]** specify only one logData section per data-object query. **Example:** The following is invalid because logData is specified more than once.

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
<log uidWell="W-12" uidWellbore="B-01" uid="L001">
   <logData>
         <mnemonicList>MDepth,ROP</mnemonicList>
         <unitList></unitList>
         <data></data>
   </logData>
   <logData>
         <mnemonicList>MDepth,ecd</mnemonicList>
         <unitList></unitList>
         <data></data>
   </logData>
</log>
</logs>
```

Invalid because logData recurs.

- The clients MUST NOT **[else error -482]** specify duplicate mnemonics in a query.

If a Server Limits Returned Data

- To avoid clients overloading a server, a WITSML server MAY limit your results set when you ask for growing data-object data. When the return value is +2, the server has limited the result to a

certain number of data-nodes and the client will need to ask again for the missing data beginning at the end of the previous returned data. A client can continue to request missing data until all the nodes in the data-object have been retrieved. OptionsIn keywords maxReturnNodes and intervalRangeInclusion are discussed in Section 6.6.2.1, page 55.

- If a server limits the amount of data that can be retrieved from growing data-objects, then the server MUST define those limits in the server Capabilities Object, by setting elements maxDataNodes and maxDataPoints as appropriate for each data-object (see Section 4.2.4.1, page 30).
  - The value of maxDataNodes represents the maximum number of data-nodes that can be handled by a function (per request) for the specified data-object.
    - For a systematically growing data-object, this represents the number of data rows.
    - For a randomly growing data-object, this represents the number of sub-objects.
    - For a growing data-object with multiple sub-object (e.g., mudLog), it represents the cumulative count of those sub-objects.
  - The value of maxDataPoints represents the maximum number of points that can be handled by a function for the specified data-object. This represents “the number of columns times the number of rows” (i.e., a cell count).

How a Server Truncates Data

If a server applies maximum-size criteria (e.g., using maxReturnNodes or using server maximum data capabilities), the server will not return the entire set described above. The server will reduce or truncate the set according to the criteria described here.

- The truncation-end specifies whether larger or smaller index values will be truncated.
  - For non-ordered data-objects (e.g., trajectory, mudLog), the server truncates the larger node-index values.
  - For a single table data-object (e.g., log), when the value of element direction is "increasing", the server truncates the larger node-index values.
  - For a single table data-object (e.g., log), when the value of element direction is "decreasing", the smaller node-index values are truncated.
- For a point index, a server MUST eliminate the data-nodes from the truncation-end such that no eliminated node will have a node-index that is at or within the range of a selected-node.
  - For a truncation-end that is larger, the server MUST ensure that a truncated node has a node-index that is greater than the node-index of a (resulting) selected-node. To retrieve the remaining nodes, the client submits a subsequent query with a structural-range minimum equal to the maximum returned node-index.
  - For a truncation-end that is smaller, the server MUST ensure that a truncated node has a node-index that is less than the node-index of a (resulting) selected-node. To retrieve the remaining nodes, the client submits a subsequent query with a structural-range maximum equal to the minimum returned node-index.

Recommended Client Behavior for Managing the Volume of Returned Data

- If the amount of growing data to be retrieved might be large, then it is RECOMMENDED that clients limit a request to one data-object so that it is obvious which data-object has been truncated if the request results in a return code of +2.
- Alternatively, it is RECOMMENDED that a client limit the request to one data-object instance per query and that the client specify the OptionsIn keyword ‘maxReturnNodes' with a value that is less than the smaller of the two items listed below (This alternative ensures that the server does not truncate the results because maxReturnNodes applies per query.):
  - For systematically growing data-objects: (maxDataPoints /(“the number of queries” x “the maximum number of curves per data-object”))

- maxDataNodes / “the number of queries”

- A client SHOULD expect growing data-object (e.g., log, mudLog) data to be returned in chunks. A RECOMMENDED practice is for clients to first retrieve the header of the data-object and then retrieve the data components.
  - If the server returns a return value of +2 (indicating that some requested data was not returned), then the client should execute a new call requesting data from the end of the previous return.
  - If the data-object is not growing, then a client should monitor the element dTimLastChange or data-object changeLog to determine when changes have occurred. For more information, see Chapter 9, page 101.

6.6.5 Standard Query Templates

This section describes the standard query templates for the WMLS_GetFromStore function and the expected WITSML server response. These standard queries were selected and developed to support the most common use cases and to ensure that servers support the queries and that clients receive a response.

All WITSML servers that support the function MUST support these queries. Optionally, server developers are free to include additional query functionality (beyond these templates).

For information about general rules for WITSML templates, see Section 4.1, page 23.

Table 3 (page 67) lists and describes all standard queries and provides links to more information and examples of each in this document.

NOTE: The templates described in this section are not intended to define behavior; they use behavior defined elsewhere in this document.

Background

The WITSML API specifies a mechanism to query for data that uses the OptionsIn parameter and XML templates to specify what data items are to be returned and what data items to use for data-object selection.

A single data-object schema contains many elements and attributes, making it possible to generate millions of different queries with different combinations of data items to be retrieved and different data-object selection criteria.

Some of these combinations do not make sense, because they do not reflect real use cases, for example "give me all the logs that have a curve of type "float". Some of these combinations are not supported by a server for practical reasons, for example "give me all the logs in the system" would overload a server.

The standard queries define the main, basic use cases that are expected.

Template Overview

To better define the required query behavior, these templates use the OptionsIn returnElements parameter (for more information, see Section 6.6.2.1, page 55).

There are two types of query.

- Data-object identification data to return a list of identifiers and names, which is typically used by a client application to build a menu, tree view, or choice list (OptionsIn returnElements=id-only).
- Query that retrieves the detailed data associated with a data-object, with additional functionality for filtering sub-sections of the WITSML growing data-objects, such as trajectory, mudLog, and log.

The List of Standard Query Templates table in Section 6.6.5.1, page 67, should be read in the following way:

- If a server receives a query that complies with the following three things (which correspond to columns in the table), it must support the query and generate a non-error response (assuming no other specification violations are detected in the query, such as a bad value for a field)
  - Is for an object in the "Objects it Applies to" column
  - The returnElements OptionsIn is set to the value specified in the "OptionsIn (returnElements)" column
  - The query includes values for ALL of the elements/attributes in the "Included in template" column

Examples to Clarify the Previous Statement

- If any of the values of the elements/attributes in the "Included in template" column are missing, then the server MAY not support the query.
  For example, in “Get Details of a Specific Data-object” the uid is required. A server MAY not support returning the details of all the objects in a wellbore. However all servers need to support returning the details for a single data-object.
- If the returnElements value is different, the server MAY not support the query.
  For example, in "Get ID of all Instances of a Data-object in a Wellbore", if the returnElements is set to ALL, the server may not support the query. It is only mandated to return the list of uid/names for all the objects in a wellbore.

Notes

- The server still needs to support any other value for an element/attribute for which it supports data-object selection as defined in this specification (for required data-object selection support rules, see Chapter 7, page 97).
  For example, a server that says it supports the "country" element as data selection criteria for a well MUST support any of the well standard queries with that element.
- The server still needs to support all the other OptionsIn as defined in this specification.

6.6.5.1 List of Standard Query Templates

NOTE: For easy reference, a unique standard query (SQ) number has been assigned to each query. If a new standard query is added, it will be assigned the next sequential number and added to the end of the current list/table.

Table 3: Standard Query Templates

Standard Query Number	Query name	OptionsIn (returnElement)	Objects it Applies to	Included in template	Returned elements/attributes (And any data-object selection elements/attributes)	Description	For More Info See:
SQ-001	Get ID of all Wells	id-only	well		uid name	Returns name and ids of all of the wells in the server that meet the selection criteria.	6.6.5.2, page 70
SQ-002	Get ID of a Well	id-only	well	uid	uid name	Returns name and id of a well with a known assigned number (uid).	6.6.5.3, page 71
SQ-003	Get details for a Well	all	well	uid	All elements/attributes	Returns all the information stored for a specified well.	6.6.5.4, page 71
SQ-004	Get ID of all Wellbores	Id-only	wellbore		uidWell uid nameWell name	Returns the name and id of all of the wellbores.	6.6.5.5, page 72
SQ-005	Get ID of all Wellbores in a Well	id-only	wellbore	uidWell	uidWell uid nameWell name	Returns the name and id of all of the wellbores for a specified well.	6.6.5.6, page 73
SQ-006	Get Details for a Wellbore	all	wellbore	uidWell uid	All elements/attributes	Returns all the information stored for a specified wellbore.	6.6.5.7, page 73
SQ-007	Get Details for all Wellbores in a Well	all	wellbore	uidWell	All elements/attributes	Returns the full details of all of the wellbores that have the specified value for uidWell.	6.6.5.8, page 74
SQ-008	Get ID of all Instances of a Data-object in a Wellbore	id-only	All data-objects under wellbore	uidWell uidWellbore	uidWell uidWellbore uid nameWell namewellbore name	Returns name and id for all instances of a specified data data-object within a specified well and wellbore.	6.6.5.9, page 75

Table 3: Standard Query Templates							
Standard Query Number	**Query name**	**OptionsIn (returnElement)**	**Objects it Applies to**	**Included in template**	**Returned elements/attributes** (And any data-object selection elements/attributes)	**Description**	**For More Info See:**
SQ-009	Get Details of a Specific Data-object	all	All data-objects under wellbore	uidWell uidWellbore uid	All elements/attributes	Returns all the information stored for a specified data-object within a specified well and wellbore.	6.6.5.10, page 76
SQ-010	Get Header for one Growing Data-object	header-only	trajectory mudLog log	uidWell uidWellbore uid	Returns: • trajectory: All data except the “trajectoryStation” elements. • mudLog: All data except the "parameter" and “geologyInterval” elements. • log: All data except the “logData” element	Returns the header for one growing data data-object.	6.6.5.11, page 78
SQ-011	Get Header for all Growing Data-objects in a Wellbore	header-only	trajectory mudLog log	uidWell uidWellbore	Returns: • trajectory: All data except the “trajectoryStation” elements. • mudLog: All data except the "parameter" and “geologyInterval” elements. • log: All data except the logData element	Returns the header of all growing data data-objects of certain type in a wellbore.	6.6.5.12, page 78
SQ-012	Get Data for Growing Data-object	data-only	trajectory mudLog	uidWell uidWellbore uid	Minimal header trajectoryStation parameter geologyInterval	Returns the selected data of a growing data data-object plus relevant non-growing information.	6.6.5.13, page 80
SQ-013	Get Restricted Data Subset	station-location-only	Trajectory	uidWell uidWellbore uid	Minimal header dTimStn, typeTrajStation, md, tvd, incl, azi, location	Returns a specified set of elements for trajectory stations.	6.6.5.14, page 81
SQ-014	Get Log Data Subset	data-only	log	uidWell uidWellbore uid	Minimal header logData	Returns the selected data of a log data-object.	6.6.5.15, page 81

Table 3: Standard Query Templates							
Standard Query Number	**Query name**	**OptionsIn (returnElement)**	**Objects it Applies to**	**Included in template**	**Returned elements/attributes** (And any data-object selection elements/attributes)	**Description**	**For More Info See:**
SQ-015	Get Log	all	log	uidWell uidWellbore uid	All elements/attributes	Returns a full log data-object.	6.6.5.16, page 82
SQ-016	What has Changed since a specified time	latest-change-only	changeLog	dTimLastChange	All elements/attributes except changeHistory	Returns information about all changes made to all data-objects on the server since a specified time.	6.6.5.17, page 84
SQ-017	What has been added to the server since a specified time	all	changeLog	changeType=add dTimChange	All elements/attributes including changeHistory	Returns the data-objects that were created since a specified time.	6.6.5.18, page 85
SQ-018	What changes have been made in a log/mudLog or multiple logs/mudLogs since a specified time	all	changeLog	objectType=log or mudLog dTimLastChange dTimChange uidWell	All changes including changeHistory for the particular log/mudLog or multiple logs/mudLogs	Returns the logs or mudLogs that have been updated since a specified time, including their changeHistory.	6.6.5.19, page 86

6.6.5.2 Get ID of All Wells (SQ-001)

Purpose of this Query: If no data-object selection criteria are specified, returns a list of the uids and names for all wells on the server). Or, returns a list of uids and names for the wells that match the data-object selection criteria specified.

The query must specify the singular data-object even though there are no selection-criteria.

The client must not specify one or more additional data-object selection-criteria elements unless they are declared as supported by the server (see Section 6.6.2.1, page 55).

Example Query (without data-object selection criteria)

This example query returns a list of the id and name for all wells on the server.

OptionsIn	`returnElements=id-only`

QueryIn Example	`<wells xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">`
	`  <well />`
	`</wells>`

XMLout

The following example results indicate that there are only four wells on the server.

```
<wells xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
   <well uid="well1">
      <name>6507/7-A-42</name>
   </well>
   <well uid="well2">
      <name>6507/7-A-43</name>
   </well>
   <well uid="well3">
      <name>6507/7-A-44</name>
   </well>
   <well uid="well4">
      <name>6507/7-A-45</name>
   </well>
</wells>
```

Example Query (with data-object selection criteria)

This example query returns a list of the id and name for all wells from “Norway” (assuming the server supports object selection using the <country> element).

OptionsIn	`returnElements=id-only`

QueryIn Example	`<wells xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">`
	`  <well>`
	`  <country>Norway</country>`
	`  </well>`
	`</wells>`

XMLout

The following example results indicate that there are only two wells on the server with country set to “Norway”.

```
<wells xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
       <well uid="well1">
              <name>6507/7-A-42</name>
       <country>Norway</country>
       </well>
       <well uid="well3">
              <name>6507/7-A-44</name>
       <country>Norway</country>
       </well>
</wells>
```

6.6.5.3 Get ID of a Well (SQ-002)

Purpose of this Query: To get the name and id of a well with a known uid value

Example Query

This query requests the uid and name of well with uid of “UUID-1”. Unless there are data problems, the server should return only one well.

OptionsIn	`returnElements=id-only`

QueryIn Example	`<wells xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">`
	`  <well uid="UUID-1" />`
	`</wells>`

XMLout

```
<wells xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
   <well uid="UUID-1">
      <name>6507/7-A-42</name>
   </well>
</wells>
```

6.6.5.4 Get Details for a Well (SQ-003)

Purpose of this Query: Requests the full details of the well data-object that matches the specified uid value. The match is not case-sensitive, so queries for “aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa” and “AAAAAAAA-aaaa-aaaa-aaaa-aaaaaaaaaaaa” yield the same results.

Example Query

OptionsIn	`returnElements=all`

QueryIn Examplenew	`<wells xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">`
	`  <well uid="aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa" />`
	`</wells>`

XMLout

```
<wells xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
 <well uid="aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa">
  <name>6507/7-A-42</name>
  <nameLegal>Company Legal Name</nameLegal>
  <numLicense>Company License Number</numLicense>
  <numGovt>Govt-Number</numGovt>
  <dTimLicense>2001-05-15T13:20:00.000Z</dTimLicense>
```

```
  …
  …Other well elements in here
  …
  <wellCRS uid="localWell1">
   <name>WellOneWSP</name>
   <localCRS>
    <usesWellAsOrigin>true</usesWellAsOrigin>
    <yAxisAzimuth uom="dega" northDirection="grid north">0</yAxisAzimuth>
    <xRotationCounterClockwise>false</xRotationCounterClockwise>
   </localCRS>
  </wellCRS>
  <commonData>
   <dTimCreation>2001-04-30T08:15:00.000Z</dTimCreation>
   <dTimLastChange>2001-05-31T08:15:00.000Z</dTimLastChange>
   <itemState>plan</itemState>
   <comments>These are the comments associated with the Well data-object.</comments>
   <defaultDatum uidRef="KB">Kelly Bushing</defaultDatum>
  </commonData>
 </well>
</wells>
```

6.6.5.5 Get ID of All Wellbores (SQ-004)

Purpose of this Query: To get the name and id of all wellbores available for the WITSML Client user

Example Query

Get all wellbores available for the WITSML client user.

OptionsIn	`returnElements=id-only`

QueryIn Example	`<?xml version="1.0" encoding="utf-8" standalone="yes"?>`
	`<wellbores xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">`
	`  <wellbore uidWell="" uid="">`
	`  </wellbore>`
	`</wellbores>`

XMLout

```
<?xml version="1.0" encoding="UTF-8"?>
<wellbores xmlns=http://www.witsml.org/schemas/1series
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.4.1.1">
   <wellbore uidWell="W1" uid="569c274d-2e90-4f31-88a0-eb4fa68a4bc2">
      <nameWell>Well 1</nameWell>
      <name>Wellbore 1</name>
   </wellbore>
   <wellbore uidWell="W2" uid="469c274d-2e90-4f31-88a0-eb4fa68a4bc3">
      <nameWell>Well 2</nameWell>
      <name>Wellbore2</name>
   </wellbore>
   <wellbore uidWell="W3" uid="469c274d-2e90-4f31-88a0-eb4fa68a4bc4">
      <nameWell>Well 3</nameWell>
      <name>Wellbore 3</name>
   </wellbore>
</wellbores>
```

6.6.5.6 Get ID of All Wellbores in a Well (SQ-005)

Purpose of this Query: Requests the nameWell, uid and name for all wellbores that have the specified uidWell value.

Example Query

OptionsIn	`returnElements=id-only`

QueryIn Example	`<wellbores xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">`
	`  <wellbore uidWell="UUID-1" />`
	`</wellbores>`

XMLout

```
<wellbores xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
   <wellbore uidWell="UUID-1" uid="B-01">
      <nameWell>6507/7-A-42</nameWell>
      <name>A-42</name>
   </wellbore>
   <wellbore uidWell="UUID-1" uid="B-02">
      <nameWell>6507/7-A-42</nameWell>
      <name>A-42 T2</name>
   </wellbore>
</wellbores>
```

6.6.5.7 Get Details for a Wellbore (SQ-006)

Purpose of this Query: Requests the full details of the wellbore data-object that matches the specified `uid` value. The match is not case-sensitive, so queries for well “UUID-1” and “uuid-1” or wellbore “b-01” and “B-01” yield the same results.

Example Query

OptionsIn	`returnElements=all`

QueryIn Example	`<wellbores xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">`
	`  <wellbore uidWell="UUID-1" uid="B-01"/>`
	`</wellbores>`

XMLout

```
<wellbores xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
  <wellbore uidWell="UUID-1" uid="B-01">
      <nameWell>6507/7-A-42</nameWell>
      <name>A-42</name>
      <number>1234-0987</number>
      <suffixAPI>02</suffixAPI>
      <numGovt>Govt001</numGovt>
      <statusWellbore>active</statusWellbore>
      <purposeWellbore>exploration</purposeWellbore>
      <typeWellbore>initial</typeWellbore>
      <shape>horizontal</shape>
      <dTimKickoff>2001-03-15T13:20:00.000Z</dTimKickoff>
      <md uom="ft">0</md>
      <tvd uom="ft">0</tvd>
```

```
      <mdKickoff uom="ft">0</mdKickoff>
      <tvdKickoff uom="ft">0</tvdKickoff>
      <mdPlanned uom="ft">15800</mdPlanned>
      <tvdPlanned uom="ft">12567</tvdPlanned>
      <mdSubSeaPlanned uom="ft">12800</mdSubSeaPlanned>
      <tvdSubSeaPlanned uom="ft">9567</tvdSubSeaPlanned>
      <dayTarget uom="d">128</dayTarget>
      <commonData>
         <dTimCreation>2001-04-30T08:15:00.000Z</dTimCreation>
         <dTimLastChange>2001-05-31T08:15:00.000Z</dTimLastChange>
         <itemState>plan</itemState>
         <comments>Comments associated with the Wellbore.</comments>
      </commonData>
   </wellbore>
</wellbores>
```

6.6.5.8 Get Details for all Wellbores in a Well (SQ-007)

Purpose of this Query: Requests the full details of all of the wellbores that have the specified value for `uidWell`.

Example Query

An identical pattern will apply for any other data-objects that are directly dependent on well, e.g., the attachment data-object. Just substitute the data-object name in the QueryIn string.

OptionsIn	`returnElements=all`

QueryIn Example	`<wellbores xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">`
	`  <wellbore uidWell="UUID-1" />`
	`</wellbores>`

XMLout

```
<?xml version="1.0"?>
<wellbores xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
   <wellbore uidWell="UUID-1" uid="B-01">
      <nameWell>6507/7-A-42</nameWell>
      <name>A-42</name>
      <number>1234-0987</number>
      <suffixAPI>02</suffixAPI>
      <numGovt>Govt001</numGovt>
      <statusWellbore>active</statusWellbore>
      <purposeWellbore>exploration</purposeWellbore>
      <typeWellbore>initial</typeWellbore>
      <shape>horizontal</shape>
      <dTimKickoff>2001-03-15T13:20:00.000Z</dTimKickoff>
      <md uom="ft">0</md>
      <tvd uom="ft">0</tvd>
      <mdKickoff uom="ft">0</mdKickoff>
      <tvdKickoff uom="ft">0</tvdKickoff>
      <mdPlanned uom="ft">15800</mdPlanned>
      <tvdPlanned uom="ft">12567</tvdPlanned>
      <mdSubSeaPlanned uom="ft">12800</mdSubSeaPlanned>
      <tvdSubSeaPlanned uom="ft">9567</tvdSubSeaPlanned>
      <dayTarget uom="d">128</dayTarget>
      <commonData>
         <dTimCreation>2001-04-30T08:15:00.000Z</dTimCreation>
         <dTimLastChange>2001-05-31T08:15:00.000Z</dTimLastChange>
         <itemState>plan</itemState>
```

```
         <comments>Comments associated with the Wellbore.</comments>
      </commonData>
   </wellbore>
   <wellbore uidWell="UUID-1" uid="B-02">
      <nameWell>6507/7-A-42</nameWell>
      <name>A-42 T2</name>
      <number>1234-0988</number>
      <suffixAPI>02</suffixAPI>
      <numGovt>Govt001</numGovt>
      <statusWellbore>active</statusWellbore>
      <purposeWellbore>exploration</purposeWellbore>
      <typeWellbore>initial</typeWellbore>
      <shape>horizontal</shape>
      <dTimKickoff>2001-03-14T13:20:00.000Z</dTimKickoff>
      <md uom="ft">0</md>
      <tvd uom="ft">0</tvd>
      <mdKickoff uom="ft">0</mdKickoff>
      <tvdKickoff uom="ft">0</tvdKickoff>
      <mdPlanned uom="ft">14800</mdPlanned>
      <tvdPlanned uom="ft">11567</tvdPlanned>
      <mdSubSeaPlanned uom="ft">11800</mdSubSeaPlanned>
      <tvdSubSeaPlanned uom="ft">9467</tvdSubSeaPlanned>
      <dayTarget uom="d">127</dayTarget>
      <commonData>
         <dTimCreation>2001-04-29T08:15:00.000Z</dTimCreation>
         <dTimLastChange>2001-05-30T08:15:00.000Z</dTimLastChange>
         <itemState>plan</itemState>
         <comments>Comments associated with the Wellbore.</comments>
      </commonData>
   </wellbore>
</wellbores>
```

6.6.5.9 Get ID of all Instances of a Data-object in a Wellbore (SQ-008)

Purpose of this Query: Requests the ancestor-name(s), ancestor-uid(s), name and uid of all instances of a specified data-object type within the specified well and wellbore.

Example Queries

OptionsIn	`returnElements=id-only`

QueryIn Example trajectory	`<trajectorys xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">`
	`  <trajectory uidWell="UUID-1" uidWellbore="B-01" />`
	`</trajectorys>`

QueryIn Example opsReport	`<opsReports xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">`
	`  <opsReport uidWell="UUID-1" idWellbore="B-01"/>`
	`</opsReports>`

XMLout (trajectory)

```
<trajectorys
   xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
   <trajectory uidWell="UUID-1" uidWellbore="B-01" uid="pe84e">
      <nameWell>6507/7-A-42</nameWell>
      <nameWellbore>A-42</nameWellbore>
      <name>Plan #2</name>
   </trajectory>
   <trajectory uidWell="UUID-1" uidWellbore="B-01" uid="pe85a">
      <nameWell>6507/7-A-42</nameWell>
```

```
      <nameWellbore>A-42</nameWellbore>
      <name>LWD Survey</name>
   </trajectory>
   <trajectory uidWell="UUID-1" uidWellbore="B-01" uid="pe86a">
      <nameWell>6507/7-A-42</nameWell>
      <nameWellbore>A-42</nameWellbore>
      <name>Gyro Survey</name>
   </trajectory>
</trajectorys>
```

XMLout (OpsReport)

```
<opsReports xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
   <opsReport uidWell="UUID-1" uidWellbore="B-01" uid="h45a">
      <nameWell>6507/7-A-42</nameWell>
      <nameWellbore>A-42</nameWellbore>
      <name>19Jun2010</name>
   </opsReport>
   <opsReport uidWell="UUID-1" uidWellbore="B-01" uid="h45b">
      <nameWell>6507/7-A-42</nameWell>
      <nameWellbore>A-42</nameWellbore>
      <name>20Jun2010</name>
   </opsReport>
   <opsReport uidWell="UUID-1" uidWellbore="B-01" uid="h45c">
      <nameWell>6507/7-A-42</nameWell>
      <nameWellbore>A-42</nameWellbore>
      <name>21Jun2010</name>
   </opsReport>
</opsReports>
```

6.6.5.10 Get Details for a Specific Data-object (SQ-009)

Purpose of this Query: Returns a single instance of the data-object that contains all of its sub-elements that are available in the server. This query template applies to all data-objects that are dependent on the wellbore. The uids for the well, the wellbore and the data-object are specified and the OptionsIn parameter returnElements is set to all.

Example Queries

An identical pattern applies for all data-objects that are dependent on wellbore. Just substitute the data-object name for trajectory or opsReport in these examples.

OptionsIn	`returnElements=all`

QueryIn Example trajectorys	Returns the full details for the specified single trajectory data-object.
	`<trajectorys xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">`
	`  <trajectory uidWell="UUID-1" uidWellbore="B-01" uid="pe84e"/>`
	`</trajectorys>`

QueryIn Example opsReport	`<opsReports xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">`
	`  <opsReport uidWell="UUID-1" uidWellbore=""B-01" uid="pe84e"/>`
	`</opsReports>`

XMLout (trajectory)

```
<trajectorys xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
   <trajectory uidWell="UUID-1" uidWellbore="B-01" uid="pe84e">
      <nameWell>6507/7-A-42</nameWell>
      <nameWellbore>A-42</nameWellbore>
```

```
      <name>Plan #2</name>
      <dTimTrajStart>2001-10-31T08:15:00.000Z</dTimTrajStart>
      <dTimTrajEnd>2001-11-03T16:30:00.000Z</dTimTrajEnd>
      <mdMn uom="ft">5432.8</mdMn>
      <mdMx uom="ft">14089.3</mdMx>
      <serviceCompany>Anadrill</serviceCompany>
      <magDeclUsed uom="dega">-4.038</magDeclUsed>
      <gridCorUsed uom="dega">0.99961</gridCorUsed>
      <aziVertSect uom="dega">82.700</aziVertSect>
      <dispNsVertSectOrig uom="ft">0</dispNsVertSectOrig>
      <dispEwVertSectOrig uom="ft">0</dispEwVertSectOrig>
      <definitive>true</definitive>
      <memory>true</memory>
      <finalTraj>true</finalTraj>
      <aziRef>grid north</aziRef>
      <trajectoryStation uid="34ht5">
         <dTimStn>2001-10-21T08:15:00.000Z</dTimStn>
         <typeTrajStation>unknown</typeTrajStation>
         <md uom="ft">5673.5</md>
         <tvd uom="ft">5432.8</tvd>
         <incl uom="dega">12.4</incl>
         <azi uom="dega">47.3</azi>
         <mtf uom="dega">47.3</mtf>
         <gtf uom="dega">0</gtf>
         <dispNs uom="ft">0</dispNs>
         <dispEw uom="ft">0</dispEw>
         <vertSect uom="ft">0</vertSect>
         <dls uom="dega/ft">0</dls>
         <rateTurn uom="dega/ft">0</rateTurn>
         <rateBuild uom="dega/ft">0</rateBuild>
         <mdDelta uom="ft">0</mdDelta>
         <tvdDelta uom="ft">0</tvdDelta>
         <modelToolError>good gyro</modelToolError>
         <gravTotalUncert uom="m/s2">0</gravTotalUncert>
         <dipAngleUncert uom="dega">0</dipAngleUncert>
         <magTotalUncert uom="nT">0</magTotalUncert>
         <gravAccelCorUsed>false</gravAccelCorUsed>
         <magXAxialCorUsed>false</magXAxialCorUsed>
         <sagCorUsed>false</sagCorUsed>
         <magDrlstrCorUsed>false</magDrlstrCorUsed>
         <statusTrajStation>position</statusTrajStation>
         <rawData>
            <gravAxialRaw uom="ft/s2">0.116</gravAxialRaw>
            <gravTran1Raw uom="ft/s2">-0.168</gravTran1Raw>
            <gravTran2Raw uom="ft/s2">-1654</gravTran2Raw>
            <magAxialRaw uom="nT">22.77</magAxialRaw>
            <magTran1Raw uom="nT">22.5</magTran1Raw>
            <magTran2Raw uom="nT">27.05</magTran2Raw>
         </rawData>
      </trajectoryStation>
      (more stations)
      <commonData>
         <itemState>plan</itemState>
         <comments>Comments for the trajectory.</comments>
      </commonData>
   </trajectory>
</trajectorys>
```

XMLout (opsReport)

The output for this example is very long so it has been moved to an APPENDIX D: Examples. See Section 14.3, page 141.

6.6.5.11 Get Data for Non-growing Part (Header) of a Growing Data-object (trajectory, mudLog, log) (SQ-010)

Purpose of this Query: Returns the header data for one of the growing data-objects. The header includes all of the data that is not a part of the growing section of the log, including common data and custom data.

- Trajectory: All data, except the trajectoryStation elements.
- MudLog: All data, except the parameter and geologyInterval elements.
- Log: All data, except the logData element.

Example Query

An identical pattern applies for all growing data-objects that are dependent on wellbore. Just substitute the data-object name in the QueryIn string and substitute the data-object uid value.

OptionsIn	`returnElements=header-only`

QueryIn Example trajectorys	`<trajectorys xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">`
	`  <trajectory uidWell=UUID-1" uidWellbore="B-01" uid="pe84"/>`
	`</trajectorys>`

XMLout

Here is the data that might be returned for the trajectory header query.

```
<trajectorys xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
   <trajectory uidWell="UUID-1" uidWellbore="B-01" uid="pe84e">
      <nameWell>6507/7-A-42</nameWell>
      <nameWellbore>A-42</nameWellbore>
      <name>Plan #2</name>
      <dTimTrajStart>2001-10-31T08:15:00.000Z</dTimTrajStart>
      <dTimTrajEnd>2001-11-03T16:30:00.000Z</dTimTrajEnd>
      <mdMn uom="ft">0</mdMn>
      <mdMx uom="ft">14089.3</mdMx>
      <serviceCompany>Anadrill</serviceCompany>
      <magDeclUsed uom="dega">-4.038</magDeclUsed>
      <gridCorUsed uom="dega">0.99961</gridCorUsed>
      <aziVertSect uom="dega">82.700</aziVertSect>
      <dispNsVertSectOrig uom="ft">0</dispNsVertSectOrig>
      <dispEwVertSectOrig uom="ft">0</dispEwVertSectOrig>
      <definitive>true</definitive>
      <memory>true</memory>
      <finalTraj>true</finalTraj>
      <aziRef>grid north</aziRef>
      <commonData>
         <itemState>plan</itemState>
         <comments>Comments for the trajectory.</comments>
      </commonData>
   </trajectory>
</trajectorys>
```

6.6.5.12 Get Header for all Growing data-objects in a Wellbore (SQ-011)

Purpose of this Query: This query example is very similar to the previous query (Section 0) for a single growing data-object header except that the uid for the data-object is not specified. The query returns the header data for all instances of the data-object in the specified well and wellbore.

An identical pattern applies for the other growing data-objects. Just substitute the data-object name (mudLog or log) for “trajectory” in the query.

Example Query

OptionsIn	`returnElements=header-only`

QueryIn Example trajectorys	`<trajectorys xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">`
	`  <trajectory uidWell="UUID-1" uidWellbore="B-01" />`
	`</trajectorys>`

XMLout

Here is the data that might be returned for trajectory.

```
<trajectorys xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
   <trajectory uidWell="UUID-1" uidWellbore="B-01" uid="pe84e">
      <nameWell>6507/7-A-42</nameWell>
      <nameWellbore>A-42</nameWellbore>
      <name>Plan #2</name>
      <dTimTrajStart>2001-10-31T08:15:00.000Z</dTimTrajStart>
      <dTimTrajEnd>2001-11-03T16:30:00.000Z</dTimTrajEnd>
      <mdMn uom="ft">0</mdMn>
      <mdMx uom="ft">14089.3</mdMx>
      <serviceCompany>Anadrill</serviceCompany>
      <magDeclUsed uom="dega">-4.038</magDeclUsed>
      <gridCorUsed uom="dega">0.99961</gridCorUsed>
      <aziVertSect uom="dega">82.700</aziVertSect>
      <dispNsVertSectOrig uom="ft">0</dispNsVertSectOrig>
      <dispEwVertSectOrig uom="ft">0</dispEwVertSectOrig>
      <definitive>true</definitive>
      <memory>true</memory>
      <finalTraj>false</finalTraj>
      <aziRef>grid north</aziRef>
      <commonData>
         <itemState>plan</itemState>
         <comments>Comments for the trajectory.</comments>
      </commonData>
   </trajectory>
   <trajectory uidWell="UUID-1" uidWellbore="B-01" uid="pe85e">
      <nameWell>6507/7-A-42</nameWell>
      <nameWellbore>A-42</nameWellbore>
      <name>Plan #3</name>
      <dTimTrajStart>2001-10-31T09:00:00.000Z</dTimTrajStart>
      <dTimTrajEnd>2001-11-03T18:12:00.000Z</dTimTrajEnd>
      <mdMn uom="ft">15000</mdMn>
      <mdMx uom="ft">20123.5</mdMx>
      <serviceCompany>Anadrill</serviceCompany>
      <magDeclUsed uom="dega">-4.038</magDeclUsed>
      <gridCorUsed uom="dega">0.99961</gridCorUsed>
      <aziVertSect uom="dega">82.700</aziVertSect>
      <dispNsVertSectOrig uom="ft">0</dispNsVertSectOrig>
      <dispEwVertSectOrig uom="ft">0</dispEwVertSectOrig>
      <definitive>true</definitive>
      <memory>true</memory>
      <finalTraj>true</finalTraj>
      <aziRef>grid north</aziRef>
      <commonData>
         <itemState>plan</itemState>
         <comments>Comments for the trajectory.</comments>
      </commonData>
   </trajectory>
</trajectorys>
```

6.6.5.13 Get Data for Growing Data-object (SQ-012)

Purpose of this Query: Requests the data portion of a growing data-object within the specified index range. Only the selection criteria is returned from the header.

Example Queries

OptionsIn	`returnElements=data-only`

QueryIn Example Trajectorys	`<trajectorys xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">`
	`  <trajectory uidWell="UUID-1" uidWellbore="B-01" uid="pe84e">`
	`  <mdMn uom="ft">5600</mdMn>`
	`  <mdMx uom="ft">5700</mdMx>`
	`  </trajectory>`
	`</trajectorys>`

QueryIn Example mudLogs	A similar pattern applies for all randomly growing data-objects that are dependent on wellbore. Just substitute the data-object name and the query range element names.
	`<mudLogs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">`
	`  <mudLog uidWell="UUID-1" uidWellbore="B-01" uid="pe84">`
	`  <startMd uom="ft">2000</startMd>`
	`  <endMd uom="ft">3629</endMd>`
	`  </mudLog>`
	`</mudLogs>`

XMLout (trajectorys)

```
<trajectorys xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
   <trajectory uidWell="UUID-1" uidWellbore="B-01" uid="pe84e">
      <mdMn uom="ft">5673.5</mdMn>
      <mdMx uom="ft">5699.1</mdMx>
      <trajectoryStation uid="34ht5">
         <dTimStn>2001-10-21T08:15:00.000Z</dTimStn>
         <typeTrajStation>unknown</typeTrajStation>
         <md uom="ft">5673.5</md>
         <tvd uom="ft">5432.8</tvd>
         <incl uom="dega">12.4</incl>
         <azi uom="dega">47.3</azi>
         <mtf uom="dega">47.3</mtf>
         <gtf uom="dega">0</gtf>
         <dispNs uom="ft">0</dispNs>
         <dispEw uom="ft">0</dispEw>
         <vertSect uom="ft">0</vertSect>
         <dls uom="dega/ft">0</dls>
         <rateTurn uom="dega/ft">0</rateTurn>
         <rateBuild uom="dega/ft">0</rateBuild>
         <mdDelta uom="ft">0</mdDelta>
         <tvdDelta uom="ft">0</tvdDelta>
         <modelToolError>good gyro</modelToolError>
         <gravTotalUncert uom="m/s2">0</gravTotalUncert>
         <dipAngleUncert uom="dega">0</dipAngleUncert>
         <magTotalUncert uom="nT">0</magTotalUncert>
         <gravAccelCorUsed>false</gravAccelCorUsed>
         <magXAxialCorUsed>false</magXAxialCorUsed>
         <sagCorUsed>false</sagCorUsed>
         <magDrlstrCorUsed>false</magDrlstrCorUsed>
         <statusTrajStation>position</statusTrajStation>
         <rawData>
            <gravAxialRaw uom="ft/s2">0.116</gravAxialRaw>
            <gravTran1Raw uom="ft/s2">-0.168</gravTran1Raw>
            <gravTran2Raw uom="ft/s2">-1654</gravTran2Raw>
```

```
            <magAxialRaw uom="nT">22.77</magAxialRaw>
            <magTran1Raw uom="nT">22.5</magTran1Raw>
            <magTran2Raw uom="nT">27.05</magTran2Raw>
         </rawData>
      </trajectoryStation>
      ... (more stations within the range 5600ft-5700ft)
   </trajectory>
</trajectorys>
```

XMLout (mudLogs)

The output for this example is very long so it has been moved to an APPENDIX D: Examples. See Section 14.4, page 149.

6.6.5.14 Get Restricted Data Subset (SQ-013)

Purpose of this Query: Returns just the basic data—md, tvd, inc, azi and horizontal displacement—for each trajectoryStation.

Some software applications need only this basic trajectory data so that they can either re-calculate the wellpath geometry or present a quick plot overview of the trajectory.

Example Query

OptionsIn	`returnElements=station-location-only`

QueryIn Example	`<trajectorys xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">`
	`  <trajectory uidWell="UUID-1" uidWellbore="B-01" uid="pe84e">`
	`  <mdMn uom="ft">5600</mdMn>`
	`  <mdMx uom="ft">5700</mdMx>`
	`  </trajectory>`
	`</trajectorys>`

XMLout

```
<trajectorys xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
   <trajectory uidWell="UUID-1" uidWellbore="B-01" uid="pe84e">
      <mdMn uom="ft">5673.5</mdMn>
      <mdMx uom="ft">5699.1</mdMx>
      <trajectoryStation uid="34ht5">
         <dTimStn>2001-10-21T08:15:00.000Z</dTimStn>
         <typeTrajStation>unknown</typeTrajStation>
         <md uom="ft">5673.5</md>
         <tvd uom="ft">5432.8</tvd>
         <incl uom="dega">12.4</incl>
         <azi uom="dega">47.3</azi>
         <dispNs uom="ft">0</dispNs>
         <dispEw uom="ft">0</dispEw>
      </trajectoryStation>
      ... (more stations within the range 5600ft-5700ft)
   </trajectory>
</trajectorys>
```

6.6.5.15 Get Log Data-Subset (SQ-014)

Purpose of this Query: Request a sub-set of the data from a log. To limit the returned data by depth or time interval, it can specify a range of index values. It may also provide a list of mnemonics that represent a sub-set of the curves defined in the log.

Guidelines

- To subset curves, the query may specify the mnemonicList.
- A query must not specify more than one logData section.
- A query must not specify logCurveInfo/mnemonic as part of data-object selection.

Example Query

OptionsIn	`returnElements=data-only`

QueryIn Example logs	`<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">`
	`  <log uidWell="UUID-1" uidWellbore="B-01" uid="f34a">`
	`  <startIndex uom="m">499</startIndex>`
	`  <endIndex uom="m">503</endIndex>`
	`  <logData>`
	`  <mnemonicList>Mdepth,Vdepth,Bit Dist,TQ on btm</mnemonicList>`
	`  </logData>`
	`  </log>`
	`</logs>`

XMLout

```
<logs xmlns="http://www.witsml.org/schemas/1series"    version="1.4.1.1">
   <log uidWell="UUID-1" uidWellbore="B-01" uid="f34a">
      <startIndex uom="m">499</startIndex>
      <endIndex uom="m">502.01</endIndex>
      <logData>
         <mnemonicList>Mdepth,Vdepth,Bit Dist,TQ on btm</mnemonicList>
         <unitList>m,m,m,kft.lbf </unitList>
         <data>499,498.99,1.25,0</data>
         <data>500.01,500,1.9,0.1</data>
         <data>501.03,501.02,2.92,0.02</data>
         <data>502.01,502,3.9,0.06</data>
      </logData>
   </log>
</logs>
```

6.6.5.16 Get Log (SQ-015)

Purpose of this Query: Get a specific log.

Guidelines

- To subset the data, a query MAY specify the start and/or end. Otherwise, the defaults are the start and end of the log (subject to data-truncation).
- To subset the curves, the query may specify the mnemonicLists.
- A query must not specify more than one logData section.
- A query must not specify logCurveInfo/mnemonic as part of data-object selection-criteria**.**

Example Query

OptionsIn	`returnElements=all`

QueryIn Example	`<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">`
	`  <log uidWell="UUID-1" uidWellbore="B-01" uid="f34a">`
	`  <startIndex uom="m">1003.5</startIndex>`
	`  <startIndex uom="m">1100.8</startIndex>`
	`  <logData>`
	`  <mnemonicList>Mdepth,Vdepth,Bit Dist,TQ on btm</mnemonicList>`
	`  </logData>`
	`  </log>`
	`</logs>`

XMLout

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1"
        <log uidWell="UUID-1" uidWellbore="B-01" uid="f34a">
                <nameWell>6507/7-A-42</nameWell>
                <nameWellbore>A-42</nameWellbore>
                <name>L001</name>
                <serviceCompany>Baker Hughes INTEQ</serviceCompany>
                <runNumber>12</runNumber>
                <creationDate>2001-06-18T13:20:00.000Z</creationDate>
                <description>Drilling Data Log</description>
                <indexType>measured depth</indexType>
                <startIndex uom="m">1003.5</startIndex>
                <endIndex uom="m">1100.8</endIndex>
                <stepIncrement uom="m">0</stepIncrement>
                <direction>increasing</direction>
                <indexCurve>Mdepth</indexCurve>
                <nullValue>-999.25</nullValue>
                <logParam uid="1" index="1" name="MRES" uom="ohm.m" description="Mud
Resistivity">1.25</logParam>
                <logParam uid="2" index="2" name="BDIA" uom="in" description="Bit
Diameter">12.25</logParam>
                <logCurveInfo uid="lci-1">
                        <mnemonic>Mdepth</mnemonic>
                        <classWitsml>measured depth of hole</classWitsml>
                        <unit>m</unit>
                        <mnemAlias>md</mnemAlias>
                        <nullValue>-999.25</nullValue>
                        <minIndex uom="m">1003.5</minIndex>
                        <maxIndex uom="m">1100.8</maxIndex>
                        <curveDescription>Measured depth</curveDescription>
                        <sensorOffset uom="m">0</sensorOffset>
                        <traceState>raw</traceState>
                        <typeLogData>double</typeLogData>
                </logCurveInfo>
                <logCurveInfo uid="lci-2">
                        <mnemonic>Vdepth</mnemonic>
                        <classWitsml>TVD of hole</classWitsml>
                        <unit>m</unit>
                        <mnemAlias>tvd</mnemAlias>
                        <nullValue>-999.25</nullValue>
                        <minIndex uom="m">1003.5</minIndex>
                        <maxIndex uom="m">1100.8</maxIndex>
                        <curveDescription>Vertical depth</curveDescription>
                        <sensorOffset uom="m">0</sensorOffset>
                        <traceState>raw</traceState>
                        <typeLogData>double</typeLogData>
                </logCurveInfo>
                <logCurveInfo uid="lci-3">
                        <mnemonic>Bit Dist</mnemonic>
                        <classWitsml>measured depth of DST bottom</classWitsml>
                        <unit>m</unit>
                        <mnemAlias>distBit</mnemAlias>
                        <nullValue>-999.25</nullValue>
                        <minIndex uom="m">1003.5</minIndex>
                        <maxIndex uom="m">1100.8</maxIndex>
                        <curveDescription>Distance drilled by bit</curveDescription>
                        <sensorOffset uom="m">0</sensorOffset>
                        <traceState>raw</traceState>
                        <typeLogData>double</typeLogData>
                </logCurveInfo>
                <logCurveInfo uid="lci-4">
                        <mnemonic>TQ on btm</mnemonic>
                        <classWitsml>torque (average)</classWitsml>
                        <unit>kft.lbf</unit>
                        <mnemAlias>tqOnBot</mnemAlias>
                        <nullValue>-999.25</nullValue>
```

```
                        <minIndex uom="m">1003.5</minIndex>
                        <maxIndex uom="m">1100.8</maxIndex>
                        <curveDescription>On bottom torque</curveDescription>
                        <sensorOffset uom="m">0</sensorOffset>
                        <traceState>raw</traceState>
                        <typeLogData>double</typeLogData>
                </logCurveInfo>
                <logData>
                        <mnemonicList>Mdepth,Vdepth,Bit Dist,TQ on btm</mnemonicList>
                        <unitList>m,m,m,kft.lbf</unitList>
                        <data>1003.5,498.99,1.25,0</data>
                        <data>1010.0,500,1.9,0.01</data>
                        <data>1020.0,501.02,2.92,0.02</data>
                        <data>1030.0,502,3.9,0.06</data>
                        <data>1040.0,503,4.9,0.11</data>
                        <data>1050.0,504.04,5.94,0.18</data>
                        <data>1060.0,505.00,612.03,1.83</data>
                        <data>1070.0,505.95,613.04,1.9</data>
                        <data>1080.0,506.91,614.04,1.97</data>
                        <data>1090.0,507.84,615.01,2</data>
                        <data>1100.8,508.75,616.01,2.08</data>
                </logData>
                <commonData>
                        <dTimCreation>2003-11-24T08:15:00.000Z</dTimCreation>
                        <dTimLastChange>2003-11-24T08:17:00.000Z</dTimLastChange>
                        <itemState>plan</itemState>
                        <comments>These are the comments associated with the log
object.</comments>
                </commonData>
        </log>
</logs>
```

6.6.5.17 What has changed since a specified time?(SQ-016)

Purpose of this query: Returns objects that have changed since the specified time.

For information about the changeLog and special change elements referred to in this template, see Chapter 9, page 101.

Example Query

OptionsIn	`returnElements=latest-change-only`

QueryIn Example	`<changeLogs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">`
	`  <changeLog>`
	`  <commonData>`
	`  <dTimLastChange>2010-10-28T09:44:29.859Z</dTimLastChange>`
	`  </commonData>`
	`  </changeLog>`
	`</changeLogs>`

XMLout

```
<changeLogs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
   <changeLog uidWell="UUID-1" uidWellbore="B-01" uidObject="Stian">
      <nameWell>DemoWell2</nameWell>
      <nameWellbore>Wellbore4</nameWellbore>
      <nameObject>ChangeLogTest</nameObject>
      <objectType>log</objectType>
      <lastChangeType>update</lastChangeType>
      <lastChangeInfo>Updated item</lastChangeInfo>
      <commonData>
         <dTimLastChange>2010-10-28T12:55:39.315Z</dTimLastChange>
      </commonData>
   </changeLog>
   <changeLog uidObject="UUID-2">
      <nameObject>DemoWell3</nameObject>
      <objectType>well</objectType>
      <sourceName>xxuu</sourceName>
      <lastChangeType>add</lastChangeType>
      <lastChangeInfo>Added Item</lastChangeInfo>
      <commonData>
         <dTimLastChange>2010-10-28T10:57:12.000Z</dTimLastChange>
      </commonData>
   </changeLog>
</changeLogs>
```

6.6.5.18 What has been added to the server since a specified time? (SQ-017)

Purpose of this query: To get a list of what has been added to a server since a specified time.

For information about the changeLog and special change elements referred to in this template, see Chapter 9, page 101.

Example Query

OptionsIn	`returnElements=all`

QueryIn Example	`<changeLogs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">`
	`<changeLog>`
	`<changeHistory>`
	`<dTimChange>2010-10-28T10:51:39Z</dTimChange>`
	`<changeType>add</changeType>`
	`</changeHistory>`
	`</changeLog>`
	`</changeLogs>`

XMLout

This example shows two objects of different types returned: One for which the last change was the creation (added) and another that was changed after it was added.

```
<changeLogs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
   <changeLog uidWell="UUID-1" uidWellbore="B-01" uidObject="Stian">
      <nameWell>DemoWell2</nameWell>
      <nameWellbore>Wellbore4</nameWellbore>
      <nameObject>ChangeLogTest</nameObject>
      <objectType>log</objectType>
      <lastChangeType>update</lastChangeType>
      <changeHistory>
         <dTimChange>2010-10-28T11:03:00.000Z</dTimChange>
         <changeType>add</changeType>
      </changeHistory>
      <commonData>
```

```
         <dTimLastChange>2010-10-28T12:55:39.315Z</dTimLastChange>
      </commonData>
   </changeLog>
   <changeLog uidObject="UUID-2">
      <nameObject>DemoWell3</nameObject>
      <objectType>well</objectType>
      <lastChangeType>add</lastChangeType>
      <changeHistory>
         <dTimChange>2010-10-28T10:57:12.000Z</dTimChange>
         <changeType>add</changeType>
         <changeInfo>New well created</changeInfo>
      </changeHistory>
      <commonData>
         <dTimLastChange>2010-10-28T10:57:12.000Z</dTimLastChange>
      </commonData>
   </changeLog>
</changeLogs>
```

6.6.5.19 What changes have been made in a log/mudLog or multiple logs/mudLogs since a specified time? (SQ-018)

Purpose of this query: To get a list a log or mudLog or multiple logs/mudLogs that have changed since a specified time.

For information about the changeLog and special change elements referred to in this template, see Chapter 9, page 101.

Example Query

OptionsIn	`returnElements=all`

QueryIn Example	`<?xml version="1.0" encoding="UTF-8"?>`
	`<changeLogs xmlns:witsml="http://www.witsml.org/schemas/1series"`
	`version="1.4.1.1">`
	`  <changeLog uidWell="well1">`
	`  <objectType>log</objectType>`
	`  <changeHistory>`
	`  <dTimChange>2011-08-01T12:00:00.000Z</dTimChange>`
	`  </changeHistory>`
	`  <commonData>`
	`  <dTimLastChange>2011-08-01T12:00:00.000Z</dTimLastChange>`
	`  </commonData>`
	`  </changeLog>`
	`</changeLogs>`

XMLout

```
<?xml version="1.0"?>
<changeLogs version="1.4.1.1" xmlns="http://www.witsml.org/schemas/1series">
   <changeLog uidObject="l_MD" uidWellbore="wellbore1" uidWell="well1">
      <nameWell>6507/7-A-42</nameWell>
      <nameWellbore>6507/7-A-42</nameWellbore>
      <nameObject>depthlog</nameObject>
      <objectType>log</objectType>
      <lastChangeType>update</lastChangeType>
      <changeInfo>Deleting data above seafloor</changeInfo>
      <changeHistory>
         <dTimChange>2011-08-01T12:41:00.000Z</dTimChange>
         <changeType>update</changeType>
         <changeInfo>Wrong unit was used in depth interval on acquisition
                system</changeInfo>
         <startIndex uom="m">1000</startIndex>
         <endIndex uom="m">1100</endIndex>
         <mnemonics>TQA</mnemonics>
```

```
      </changeHistory>
      <changeHistory>
         <dTimChange>2011-08-01T12:42:00.000Z</dTimChange>
         <changeType>update</changeType>
         <changeInfo>Corrupt data for curve in depth interval. Non-
                recoverable.</changeInfo>
         <startIndex uom="m">200</startIndex>
         <endIndex uom="m">400</endIndex>
         <mnemonics>GR</mnemonics>
      </changeHistory>
      <changeHistory>
         <dTimChange>2011-08-01T12:46:00.000Z</dTimChange>
         <changeType>update</changeType>
         <changeInfo>Deleting data above seafloor</changeInfo>
         <startIndex uom="m">0</startIndex>
         <endIndex uom="m">200</endIndex>
      </changeHistory>
      <commonData>
         <dTimCreation>2011-07-30T13:00:00.000Z</dTimCreation>
         <dTimLastChange>2011-08-01T12:46:00.000Z</dTimLastChange>
      </commonData>
   </changeLog>
   <changeLog uidObject="l_time" uidWellbore="wellbore1" uidWell="well1">
      <nameWell>6507/7-A-42</nameWell>
      <nameWellbore>6507/7-A-42</nameWellbore>
      <nameObject>timelog</nameObject>
      <objectType>log</objectType>
      <lastChangeType>update</lastChangeType>
      <changeInfo>Adding time based gamma</changeInfo>
      <changeHistory>
          <dTimChange>2011-08-01T12:10:00.000Z</dTimChange>
          <changeType>update</changeType>
          <changeInfo>Data was uploaded using wrong unit in time interval
                 Corrected data published.</changeInfo>
          <startDateTimeIndex>2011-08-01T09:30:00.000Z</startDateTimeIndex>
          <endDateTimeIndex>2011-08-01T09:45:00.000Z</endDateTimeIndex>
          <mnemonics>HKLD,WOB</mnemonics>
      </changeHistory>
      <changeHistory>
         <dTimChange>2011-08-01T12:25:00.000Z</dTimChange>
         <changeType>update</changeType>
         <changeInfo>No returns available in time interval, so there is no gas
                    data.</changeInfo>
         <startDateTimeIndex>2011-08-01T10:00:00.000Z</startDateTimeIndex>
         <endDateTimeIndex>2011-08-01T11:00:00.000Z</endDateTimeIndex>
         <mnemonics>TOTGAS</mnemonics>
      </changeHistory>
      <changeHistory>
         <dTimChange>2011-08-01T12:50:00.000Z</dTimChange>
         <changeType>update</changeType>
         <changeInfo>Adding time based gamma</changeInfo>
         <startDateTimeIndex>2011-08-01T12:45:00.000Z</startDateTimeIndex>
         <endDateTimeIndex>2011-08-01T12:50:00.000Z</endDateTimeIndex>
         <mnemonics>GR</mnemonics>
      </changeHistory>
      <commonData>
         <dTimCreation>2011-07-27T18:00:00.000Z</dTimCreation>
         <dTimLastChange>2011-08-01T12:50:00.000Z</dTimLastChange>
      </commonData>
   </changeLog>
   <changeLog uidObject="l_ream_MD" uidWellbore="wellbore1" uidWell="well1">
      <nameWell>6507/7-A-42</nameWell>
      <nameWellbore>6507/7-A-42</nameWellbore>
      <nameObject>ream_depthlog</nameObject>
      <objectType>log</objectType>
      <lastChangeType>add</lastChangeType>
      <changeInfo>Adding reaming log</changeInfo>
      <changeHistory>
         <dTimChange>2011-08-01T12:05:00.000Z</dTimChange>
         <changeType>add</changeType>
         <changeInfo>Adding reaming log</changeInfo>
      </changeHistory>
      <commonData>
         <dTimCreation>2011-08-01T12:05:00.000Z</dTimCreation>
```

```
         <dTimLastChange>2011-08-01T12:05:00.000Z</dTimLastChange>
      </commonData>
   </changeLog>
   <changeLog uidObject="l_image_MD" uidWellbore="wellbore1" uidWell="well1">
      <nameWell>6507/7-A-42</nameWell>
      <nameWellbore>6507/7-A-42</nameWellbore>
      <nameObject>image_depthlog</nameObject>
      <objectType>log</objectType> <lastChangeType>delete</lastChangeType>
      <changeInfo>Deleting log. Log was added by mistake.</changeInfo>
      <changeHistory>
         <dTimChange>2011-08-01T12:35:00.000Z</dTimChange>
         <changeType>delete</changeType>
         <changeInfo>Deleting log. Log was added by mistake.</changeInfo>
      </changeHistory>
      <commonData>
         <dTimCreation>2011-08-01T10:00:00.000Z</dTimCreation>
         <dTimLastChange>2011-08-01T12:35:00.000Z</dTimLastChange>
      </commonData>
   </changeLog>
</changeLogs>
```

6.7 WMLS_UpdateInStore Function

Purpose: Updates one existing WITSML data-object on the server. If the data-object does not exist in the persistent data store on the server, use WMLS_AddToStore.

The WITSML data-object with the updated/added values is passed in a single XML document, containing a root plural data-object element (e.g., <wells>) enclosing one singular data-object element (e.g., <well>). The data-object to be updated is identified using the data-object type specified by WMLtypeIn plus the unique identifier(s) (UID) present in the data-object.

This pseudo code example provides an overview of the function:

```
integer = WMLS_UpdateInStore(
                                    [in] string WMLtypeIn,
                                    [in] string XMLin,
                                    [in] string OptionsIn,
                                    [in] string CapabilitiesIn,
                                    [out] string SuppMsgOut);
```

6.7.1 Return Values

When a server finishes processing a request, it returns a numeric value. A positive value indicates a success, and a negative value indicates an error. Throughout this section, return values are indicated for specific behaviors. For a complete list of values and their corresponding messages, see APPENDIX A: Defined Return Values, page 110.

6.7.2 Parameters (all required)

Parameter	Description/Purpose/Example
WMLtypeIn	• Input string. One WITSML data-object type (see the specific WITSML data schema). • A non-empty value MUST **[else error -407]** be specified.
XMLin	• Input string. The WITSML data-object to be updated. • A non-empty value MUST **[else error -408]** be specified. • The XML MUST **[else error -409]** conform to a derived update schema (no compression specified) where all elements and attributes are optional except for all uid attributes which are mandatory or MUST **[else error -426]** validate against the derived update schema after uncompressing.
OptionsIn	• For information about formatting the OptionsIn parameter, see Section 6.1.7, page 43. • Input string. • An empty string indicates that no value was given and the server MUST use the default values of all options. • The keyword of 'compressionMethod' specifies a type of compression that the client has applied to the contents of XMLin. • A server MAY support client compression. If a server supports client compression, then the server MUST specify compressionMethod in its server Capabilities Object. See Section 4.2.4, page 30. • After a client applies a compression method, the client must **[else error -479]** encode the resultant byte array into the XMLin/QueryIn string using base64Binary. Currently recognized compression methods include:

Value	Definition
None (DEFAULT)	Specifies that client is not applying compression to QueryIn.
gzip	Specifies that the client is using gzip compression.

Parameter	Description/Purpose/Example
CapabilitiesIn	• Input string. The client's Capabilities Object (capClient) to be sent to the server. For more information, see section 4.2.2, page 28. • A server MUST interpret an empty string to mean that no value was given. • The XML MUST **[else error -466]** conform to the API capClient schema.
SuppMsgOut	• Output string. Supplemental message text.

6.7.3 UID Requirements

To identify, retrieve and update WITSML data-objects, each data-object has a machine-readable unique identifier (UID). For more information on UIDs, see Section 2.2, page 15. This table lists requirements for using UIDs within this function.

UID Requirements for UpdateInStore	
Object	**Optional/Required?**
Object UID	required
Parentage UID	required
Child UID	required

6.7.4 How it Works

For the purpose of the following discussion, a container-element is an element that contains other elements (as opposed to a value). A non-container-element is an element that has a value and/or possibly an attribute but not other elements.

- A client MUST **[else error -433]** provide a data-object with the same type and unique identifier(s) that already exists in the persistent store. (If you are adding a new data-object, use WMLS_AddToStore data-object).
- A client MUST **[else error -415]** include the unique identifier (ID) of the data-object to be updated, thereby uniquely identifying only one data-object to be updated. A client MUST NOT **[else error -444]** specify multiple data-objects.
- The client MUST **[else error -401]** provide, in the XMLin document, the plural container-element as its root XML element (e.g., wells).
- A client MUST **[else error -483]** provide an XMLin document that complies with the update schema.
- A client MUST NOT **[else -484]** apply an operation that results in a missing mandatory item. (That is, the update cannot result in missing mandatory items, in context of the write schema.)
- A server MUST support the data-object as defined in its Capabilities Object.
- A server MUST allow an element (container or not) in an existing container-element (recurring or not) to be alterable without changing anything else in the container.
- If a client sends a value of NaN in a numeric field to the server, then the server MUST convert it to null or "not given".

6.7.4.1 Update Behaviors

- A server MUST allow an attribute to be alterable without changing anything else in the element.
- If a client specifies a value for an existing non-container-element, then the server MUST replace the value for that element. No check is performed to see if the value is actually different. For example, the following example updates the itemState of well "123":

```
<wells xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
   <well uid="123">
         <commonData>
               <itemState>plan</itemState>
         </commonData>
   </well>
</wells>
```

- If a client specifies a value for an existing attribute, then the server MUST replace the value for that attribute. No check is performed to see if the value is actually different.
  - If a client identifies a new element or attribute, then the server MUST insert it.
  - A recurring container with a new UID value is a new element.
  - Each new UID value provided by a client MUST **[else error -464]** be unique within the context of its nearest recurring parent node.
  - Clients MUST NOT **[else error -445]** specify new elements or attributes that are empty. For example, the following example inserts a new trajectoryStation, assuming that it did not already exist: If it already existed, this code would change the value of md.

```
<trajectorys xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
      <trajectory uidWell="W-12" uidWellbore="B-01" uid="pe84e">
            <trajectoryStation uid="456">
                  <md uom="ft">2012.6</md>
            </trajectoryStation>
      </trajectory>
</trajectorys>
```

6.7.4.2 UIDs

- A server MUST use a caseless compare against uid values (as it does for all other values).
- If a client specifies an element (recurring or not) in a schema with a unique identifier, then the client MUST **[else error -448]** also specify the identifier value. For example, the following attempt to change the name is invalid because a uid value is not specified for trajectoryStation:

```
<trajectorys xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
   <trajectory uidWell="W-12" uidWellbore="B-01" uid="pe84e">
         <trajectoryStation>
               <md uom="ft">20055.4</md>
         </trajectoryStation>
         <trajectoryStation uid="">
               <md uom="ft"> 20065.6</md>
         </trajectoryStation>
   </trajectory>
</trajectorys>
```

Invalid because the uid values are not specified for trajectoryStation.

- A client MUST NOT **[else error -464]** add a child element that contains a uid attribute whose value matches the uid value in an existing sibling element of the same name. That is, the uid values of recurring elements are unique within the context of their closest recurring parent node.

6.7.4.3 Units of Measure (UOM)

For more information about handling units of measure, see Chapter 8, page 99.

- The client MUST **[else error -453]** always specify the uom for all measure data.
- The client MUST **[else error -443]** specify a value of uom that matches an annotation attribute from the WITSML Units Dictionary XML file.
- A client MUST NOT **[else error -446]** specify a uom attribute unless its corresponding value is specified. This rule prevents any assumption that a client can influence the default unit of measure. **Example:** In the following, the md in the first station is not allowed but the md in the second station is OK.

```
<trajectorys xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
   <trajectory uidWell="W-12" uidWellbore="B-01" uid="pe84e">
          <trajectoryStation uid="123">
                 <md uom="ft" />
          </trajectoryStation>
          <trajectoryStation uid="123">
                 <md uom="ft">667.4</md>
          </trajectoryStation>
   </trajectory>
</trajectorys>
```

Invalid because the md value is not specified.

OK because the md value is specified.

6.7.4.4 Growing data-objects and Data Limits

For more information about growing data-objects, see Chapter 5, page 36.

- If a server limits the amount of data that can be updated in growing data-objects, then the server MUST define those limits in the server Capabilities Object by setting elements maxDataNodes and maxDataPoints as appropriate for each data-object.

  The value of maxDataNodes represents the maximum number of data-nodes that can be handled by a function for the specified data-object.

  - For a systematically growing data-object this represents the number of data rows.
  - For a randomly growing data-object this represents the number of sub-objects.
  - For a growing data-object with multiple sub-object (e.g., mudLog) it represents the cumulative count of those sub-objects.

  The value of maxDataPoints represents the maximum number of points that can be handled by a function for the specified data-object:

  - This represents the number of columns multiplied by the number of rows (i.e., a cell count).
- The client MUST NOT **[else error -456]** attempt to update more data than is allowed by the servers maxDataNodes and maxDataPoints.

6.7.4.5 Additional Information for Updating Systematically Growing Data-objects (log)

Because systematically growing data-objects have no UID, they require special handling, which this section explains.

- If a client specifies data-nodes for existing columns, then the server MUST ignore structural-range. Rather, the server MUST evaluate each column of data separately to determine the update-column-range.

  The update-column-range is the minimum and maximum index in the update data for that column.

  - If the update-Colum-range extends beyond the pre-update column-range for that column, then data will have been appended.
  - If the update-column-range does not extend beyond the pre-update column-range, then interior data will have been modified.
  - **NOTE:** Any changes to data may affect dTimLastChange and objectGrowing, as described in Chapter 9, page 101.
- The update data MUST NOT **[else error -463]** contain multiple nodes with the same index.
- If a client specifies a new column-identifier and data-nodes with values, then the server MUST add values to existing data-nodes and it MUST add new data-nodes, if appropriate.
  - A client MUST NOT **[else error -480]** add a new column at the same time it is updating an existing column. (The index column does not count as being updated.)
  - A client MUST NOT **[else error -436]** specify a structural-range with this option.

  - If a client specifics column-range values, then the server MUST ignore them.
  - If a client specifies data-nodes, then the client MUST **[else error -434]** identify column-identifier in the data-column-list.

- The server MUST treat any data-node value that matches the nullValue to be the same as if consecutive commas were specified.
- The logData section MAY recur when updating sparse data. For example, the following example modifies one value in each row for different curves. Note that logData does not have a unique identifier (UID) because this recurring behavior represents a sparse view on the actual data.

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
<log uidWell="W-12" uidWellbore="B-01" uid="L001">
   <logData>
         <mnemonicList>MDepth,ROP,TQ on btm</mnemonicList>
         <unitList>m,m/h,kft.lbf</unitList>
         <data>10003,2.3,2.08</data>
   </logData>
   <logData>
         <mnemonicList>MDepth,ecd</mnemonicList>
         <unitList>m,g/cm3</unitList>
         <data>10005,1.11</data>
   </logData>
   <logData>
         <mnemonicList>MDepth,ecd</mnemonicList>
         <unitList>m,g/cm3</unitList>
         <data>10007,1.23</data>
   </logData>
</log>
</logs>
```

- When updating data in a systematically growing data-object:
  - The client MUST **[else error -449]** specify the indexCurve in the mnemonicList and the client MUST **[else error -**450] ensure that each mnemonic occur only once in the list.
  - The client MUST **[else error -451]** always specify the unitList and the client MUST **[else error -452]** also populate the list with units that match the header.

    In the above example for sparse data, the "MDepth" mnemonic represents the index curve; if it was not given, the server would return **[error code -450]**.

6.7.4.6 Log Data-objects and Clearing Behavior

This API version clarifies server behavior for updating log data-objects and related behavior for clearing existing content during an update. The examples below explain how 1.4.1 servers MUST work.

The Problem with the Previous API Version

In the previous API version, servers were doing one of the following, which resulted in inconsistent server behavior:

- Clearing the entire startIndex/endIndex range (or the startDateTimeIndex/endDateTimeIndex range) for the specified channels, OR
- Only clearing channels based upon the implicit data ranges of each log Channel.

Behavior for the Current API Version

For this latest API version, the behavior is as follow:

- When an update request is received by a server, the server does not clear the referenced channels based on the header's startIndex (or startDateTimeIndex) and endIndex (or endDateTimeIndex) values, if supplied. Instead, the server MUST use the implicit indices within

the logData section to clear only the range specified for each channel. That range is bracketed by the earliest and latest value of each channel in the update request.

- Client applications SHOULD NOT provide the startIndex and endIndex for any update requests, because the server ignores this information. The server MUST derive the index values from the update data itself.

There are two main cases to consider when updating logs:

- **Append**: The data in the Update is completely after (or completely before) any data already on the server for the log, i.e. there is NO intersection with existing data. (See Example 3 below.)
- **Intersection**: The data in the Update has *some degree* of intersection with the log's data already on the server. The server will examine its own store, on a channel by channel basis, to see if any of the data in the update overlaps. (See Examples 1 and 2 below.)

Example 1

From left to right, the following tables show: existing values on a server (Table 1), update request data (Table 2), and the update results on a v1.4.1 server, for Channel A and Channel B of a log (Table 3).

Table 1

Existing Values on the WITMSL Server

Index	Ch. A	Ch. B
1000	1	11
1001		12
1002	3	

Table 2

Update Request with Unaligned Data

Index	Ch. A	Ch. B
1002		13
1003	4	14

Table 3

Update Results on the WITMSL Server

Index	Ch. A	Ch. B
1000	1	11
1001		12
1002	3	13
1003	4	14

In the update request data (Table 2), the implicit ranges are:

- Channel A: startIndex = 1003 and endIndex = 1003
- Channel B: startIndex=1002 and endIndex=1003

The important clarification for v1.4.1 behavior is that value of Channel A at 1002 remains 3; that is, it is NOT cleared by the update—which was previously the case for some v1.3.1 servers.

This update does not update the Change Log, because it is an append operation. (For more information on the Change Log, see Section 9.3.4, page 103.)

Example 2

This example continues from the end result of Example 1. Table 3 is repeated here as the server starting value, Table 4 shows the update request data, and Table 5 the results of this update.

Table 3

Existing Values on the WITMSL Server

Index	Ch. A	Ch. B
1000	1	11
1001		12
1002	3	13
1003	4	14

Table 4

Update Request with Unaligned Data

Index	Ch. A	Ch. B
1001	5	
1002		13.5
1003	4	14.5

Table 5

Update Results on the WITMSL Server

Index	Ch. A	Ch. B
1000	1	11
1001	**5**	12
1002		**13.5**
1003	**4**	**14.5**

In the update request data (Table 2), the implicit ranges are:

- Channel A: startIndex = 1001 and endIndex = 1003
- Channel B: startIndex=1002 and endIndex=1003

The implicit range for results in the server setting the index value of 1002 to null—clearing the previously existing value of 3.

This example does modify the change log, because the server makes an interior modification to the data store. The change log would show that A was updated from 1001-1003 and B was updated from 1002-1003. (For more information on the change log, see Section 9.3.4, page 103.)

Example 3: Append

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
    <log uidWell="W-12" uidWellbore="B-01" uid="L001">
       <logData>
             <mnemonicList>Mdepth,ROP,ECD</mnemonicList>
             <unitList>ft,ft/hr,g/cm3</unitList>
             <data>5000,22.59,1.36</data>
       </logData >
    </log>
</logs>
```

Update adding one row in a particular log.

…results in the following data in the server: Note that mnemonic 'Bit RPM' was not specified in the above template, which resulted in an empty value being defined in the third column on the server. An empty value is equivalent to a null value.

Mdepth	ROP	Bit RPM	ECD
<data>4050	37.11	93.74	</data>
<data>4060	9.85	95	1.33</data>
<data>4070	32.44	89.19	1.31</data>
<data>4080	29.03		1.32</data>
<data>4090	13.09		1.34</data>
<data>5000	**22.59**		**1.36</data>**

6.7.5 Growing Data-object Examples

For more code examples, see Appendix D, Section 14.2, page 122.

6.8 WMLS_GetBaseMsg Function

Purpose: Returns a string containing only the fixed (base) message text associated with a defined Return Value (see APPENDIX A: Defined Return Values, page 110). This option can be particularly useful for custom error codes that are not in this specification.

Variable, context-specific supplemental message text and diagnostic information, if any, is returned in the SuppMsgOut parameter of the various STORE functions where it is used.

This pseudo code example provides an overview of the function:

```
string = WMLS_GetBaseMsg(
                                    [in] integer ReturnValueIn
                                     );
```

6.8.1 Parameters (all required)

Parameter	Description/Purpose/Example
ReturnValueIn	Input integer. A non-null value MUST **[else error -422]** be specified. A valid Return Value (see APPENDIX A: Defined Return Values, page 110)

6.8.2 Return Values

String. The fixed descriptive message text associated with the Return Value.

If ReturnValueIn is invalid, the server MUST return a null string.

7. Data-object Selection Support Rules

A server MUST support the following data-object items (elements or attributes) for the purpose of data-object selection.

7.1 For All Data-objects

A server MUST support the following elements/attributes for all data-object types.

Element/Attribute	Description
uid uidWell uidWellbore	All unique identifier (uid) attributes, which includes the data-object uid attribute, the data-object parentage uid attribute(s) (as listed in the first column) and all descendant-node uid attributes. **NOTE**: uidWell and uidWellbore do not exist in all data-objects, but a server MUST support them where they do exist. For the changeLog data-object, attribute uidObject represents the unique identifier. For more information about the changeLog, see Section 9.3.4, page 103.
name nameWell nameWellbore	All data-object name elements, which includes the data-object's name element and the parentage name elements (as listed in column one). NOTE: nameWell and nameWellbore do exist in all data-objects, but a server MUST support them where they do exist. For the changeLog data-object, element nameObject represents the name.
dTimLastChange	The dTimLastChange element within elements commonData and commonTime, which includes the elements commonData and commonTime within growing sub-objects. For more information about this element, see Section 9.3.3, page 103.

Guideline: A client should not use a numerical value for data-object selection.

7.2 For Specific Data-objects

In addition to the above items that a server MUST support for all data-objects, a server MUST support the following items for specific data-objects. If a server declares support for the specified data-object type, then the server MUST support the listed items for that data-object. For elements with a `uom` attribute, the server MUST also support the `uom` attribute.

Object Type	Item Name
attachment	objectReference
changeLog	nameObject
	objectType
	changeHistory/dTimChange
	changeHistory/changeType
	uidObject
log	objectGrowing
	indexType
	startIndex
	endIndex
	startDateTimeIndex
	endDateTimeIndex
	logCurveInfo/mnemonic

Object Type	Item Name
	logData/mnemonicList
message	objectReference
mudLog	objectGrowing
	startMd
	endMd
objectGroup	name
	groupType
	groupOnly
risk	objectReference
	type
	severityLevel
	probabilityLevel
trajectory	objectGrowing
	mdMn
	mdMx
well	numGovt
	numAPI

8. Units of Measure

Numeric measure values exchanged using WITSML have an associated uom attribute that describes the units of measure for that value. This approach is flexible and allows data to be exchanged with minimum constraints or assumptions.

8.1 WITSML Units Dictionary

The WITMSL standard includes the WITSML Units Dictionary file that defines the units symbols used in WITSML and contains unit conversion information.

All statements and references to units in this specification are with the understanding that the units are in the dictionary. The WITSML Units Dictionary is included in the distribution media for each WITSML Data Schema Version and is updated as part of the schema revision process.

8.2 Basic Guidelines

- The uom attribute is mandatory for all numeric values that must be qualified with a unit of measure indicator to provide the full context of an element value. The value of the uom attribute is REQUIRED to match an 'annotation' attribute from the WITSML Units Dictionary XML file. **NOTE:** This requirement does not apply to log curve unit field, though where possible it is recommended that you use a unit from the Units dictionary.

  For example, a WITSML document might contain the fragment:

```
<ropAv uom="ft/h">80.00</ropAv>
```

  This example specifies that the value for ropAv is measured in feet per hour (ft/h).

  The uom value of ft/h is the annotation for a UnitOfMeasure in the WITSML Units Dictionary reference file as:

```
<UnitOfMeasure id="ftPh" annotation="ft/h">
   <Name>feet/hour</Name>
   <CatalogName>POSC</CatalogName>
   <CatalogSymbol isExplicit="true">ft/h</CatalogSymbol>
   <ConversionToBaseUnit baseUnit="m/s">
         <Fraction>
               <Numerator>0.3048</Numerator>
               <Denominator>3600</Denominator>
         </Fraction>
   </ConversionToBaseUnit>
</UnitOfMeasure>
```

  This approach gives a link to the base SI unit of meters per second (m/s) and a suggested conversion to this base unit. In this way, any WITSML-compliant document can be decoded by any application that can work with SI units.

8.3 Server Defaults

A server MUST have a default set of units. This might be as simple as returning whatever unit is defined for the value in the persistent store and the unit might or might not be the original unit. Another default might be to return the values in a preferred unit system. How the preferred unit system for a server is set is up to the server implementation and is not part of the standard.

8.4 Server and Client Responsibilities

1. A server MAY provide units conversion (optional). This means that if a client requests data elements and specifies a uom attribute, then a server that supports unit conversion will convert the value into the requested units.

   For example, if a client requests a wellbore object and includes <md uom="ft" /> in the query, a server that supports units conversion will return the md element in feet. A server that does not support units conversion may return the md value in any valid WITSML units and will include the units in the uom attribute: <md uom="m">4855.2</md>.

   The capServer element *supportUomConversion* specifies if a server supports this behavior. (For more information about capServer elements, see Section 4.2.4.1, page 30.)

2. A server MUST honor any valid unit of measure specified by a client when adding or updating data-objects. However, this does not mean that the server will return the same unit if the client asks for that same data to be returned. The server MAY choose to convert that unit into another unit for internal use.

   For example, if a client is adding a wellbore with an md of 3,000 ft, but the server only supports meters, then the server can store the wellbore with an md of 914.4 m.

3. For a data range query on a growing data-object, a server MUST be able to do a unit conversion of the start and end index values provided by a client.

   For example, if a client wants to retrieve log data in a range of 1,000 to 2,000 meters and a server is storing the data internally in feet, then the server must convert the start and end index values into the equivalent feet values so that it can perform the query. The data may then be returned to the client with the index values in feet or meters, depending on the server's ability to perform unit conversion as described above. It is the responsibility of the client to check the units that are returned from the server.

4. A client SHOULD be written to accept any valid unit of measure returned by a server. If that unit is not appropriate for use within the client, then it is the client's responsibility to convert the unit to something else.

5. A server MUST NOT consider a unit of measure to be part of the data-object selection criteria. That is, for WMLS_GetFromStore, a server MUST ignore any unit value that is specified without a value. For example, you cannot query for all wellbores that have mdKickoff specified in feet.

6. For WMLS_AddToStore and WMLS_UpdateInStore, the client MUST **[else error -453]** always specify the unit for all measure data. The client MUST **[else error -443]** provide a value of the unit that matches an annotation attribute from the WITSML Units Dictionary XML file.

7. If client requests a unit, a server MUST always populate the value with an entry that describes the units of the element value.

8. The unit SHOULD NOT change in validated data but, because we are trying to support realtime conditions; a server is allowed to "correct" the data. If a unit changes, then the client SHOULD retrieve the modified data value because the unit is an integral part of the data value. For a curve, the client SHOULD retrieve the whole curve.

9. Whether or not a server constrains the allowed units based on back-end contractual obligations for a specific site is out of scope for this specification.

9. Mechanisms for Identifying Change in a WITSML Server

Table 1 lists the main mechanisms for indicating and identifying change for data-objects in a WITSML server. When used correctly, these mechanisms help client applications detect changes in a way that reduces client polling and server load.

This chapter describes server behavior for updating these mechanisms and how they are used.

Table 1 Main WITSML Change Mechanisms

Element/Object	Definition and Description of Use	For More Information
dTimCreation	Element contained in the commonData structure that exists in all data-objects and in the commonTime structure that exists in some growing sub-objects. Defines when the data-object or sub-object was first populated in the server.	See Section 9.3.1, page 102
dTimLastChange	Element contained in the commonData structure that exists in all data-objects and in the commonTime structure that exists in some growing sub-objects. Reflects when the most recent change has happened to any data in a particular data-object or sub-object.	See Section 9.3.3, page 103.
objectGrowing	Element contained in all growing data-objects. It is a Boolean flag that indicates whether a growing data-object is actively growing (=True) or not actively growing (=False).	See Section 9.3.2, page 102
changeLog	Special object that indicates that a change has occurred to a data-object instance in the server. The changeLog was created as a mechanism for clients to more easily detect which data-objects in a server have changed and to reduce server load. The server MUST support changeLog objects for all data-object types that it declares support for in the capServer.	See Section 9.3.4, page 103

9.1 Append: Defined

For the purpose of behavior described in this chapter, the word *append* has the meanings listed below:

- For a randomly growing object: to add a new trajectory station to a trajectory or new geology interval entry to a mud log.
- For systematically growing objects (a log): to add new curve data whose index is outside the min/max range for that curve.

For information about growing data-objects, see Chapter 5, page 36.

Conversely, the following behaviors are NOT appends.

- Adding a data-object.
- Adding a curve.
- Deleting a curve or sub-node.
- Updating the interior portion of curve (i.e., between min/max index, inclusive of end points).
- Updating a sub-node.

9.2 Change Elements from the Server Capabilities Object

The server Capabilities Object (capServer, see Section 4.2.4, page 30) MUST define values for the following elements related to detecting and identifying changes. Recommended values for these elements are outside the scope of this specification.

Element	Definition and Description of Use
changeDetectionPeriod	Defines the maximum number of seconds for the server to detect change in a growing data-object. A server MUST detect a change to any data-object within the specified time. Maximum value = 600 seconds (10 minutes) Required for growing data-objects.
growingTimeoutPeriod	If data is not appended to a growing data-object within the specified time, the server sets the objectGrowing flag for the data-object to "false". A server defines a separate time-out period for each growing data-object type that the server supports. Clients use the objectGrowing flag to determine what data-objects are actively being updated.

9.3 Change Mechanisms

This section describes the main change mechanisms and how they are updated by a server.

IMPORTANT! The mechanisms discussed in this section can only be changed by the server. The server MUST ignore any attempt by a client to change any of these parameters or objects.

Note: For Boolean items, the XML Schema standard supports use of both "true" and "1" and "false" and "0". WITSML is consistent with XML, so also supports use of both. For simplicity, the chapter refers to "true" and "false" only.

9.3.1 dTimCreation Element

dTimCreation represents the time that a data-object or sub-node was first populated in a server. This element is contained in the commonData structure that exists in all data-objects and in the commonTime structure that exists in some growing sub-objects.

For instance, an aggregating server resets dTimCreation at the time the data is loaded from a source, which may be another server or a client. If acquisition-related timestamps are relevant, a data-object will have specific elements for that information.

When a data-object or sub-node is added in a server, the server MUST set the dTimCreation to the time when the operation was detected

9.3.2 objectGrowing Element

objectGrowing is a Boolean flag that allows a client to determine which growing data-objects within that server are actively being updated. Each growing data-object has its own objectGrowing flag.

A server MUST support object selection for this field.

The server sets the flag to "true" or "false" for a specific growing data-object under certain conditions, which are described in the next section.

9.3.2.1 Logic for Setting objectGrowing Flags

For each growing data-object that has an objectGrowing flag:

- When an append occurs (as defined in Section 9.1, page 101) for a growing data-object and objectGrowing = "false", then the server sets objectGrowing = "true".
- If a server receives no data to append to a growing data-object in that data-object's growingTimeoutPeriod, then the server MUST set objectGrowing = "false" (indicating to clients that the data-object is not actively being updated).

9.3.3 dTimLastChange Element

dTimLastChange reflects when the most recent change has happened to data in a particular data-object or sub-node.

dTimLastChange is contained in the commonData structure of each data-object and in the commonTime structure that exists in some growing sub-nodes.

- The server MUST update dTimLastChange for ALL changes to a data-object.
- For non-growing data-objects, a server MUST update dTimLastChange transactionally (for each change that occurs).
- For growing data-objects, a server MUST update dTimLastChange within the specified change detection period. For example, a changeDetectionPeriod of 6 seconds means that a server is expected to update dTimLastChange within 6 seconds of the actual change to the data-object on the server.

The server MUST set the value for dTimLastChange to reflect the time of detection of the change. In an aggregating server, this is the time when that server updated the data from the source. This behavior is necessary to allow a client to query for new data since the last time it checked.

For the time to be of use across multiple machines, the internal clocks on those machines need to reference a common time source, or be synchronized in some manner. Time synchronization is outside the scope of the WITSML specification.

9.3.4 changeLog Object

The changeLog is a special server object that indicates that a change has occurred to a data-object instance in a server. Each data-object instance in a server has its own changeLog object instance.

All servers MUST:

- Register support for changeLog objects in the server's capability object (capServer, see Section 4.2.4, page 30) for the WMLS_GetFromStore function.
- Support changeLog objects for all data-object types that it declares support for in the capServer.

9.3.4.1 changeLog Elements/Attributes

Table 2 lists and defines the main changeLog elements/attributes in the changeLog schema. In most cases, the value in a changeLog instance is from a corresponding element/attribute in the associated data-object instance. The following sections describe how these elements/attributes are used to indicate changes to a data-object instance in a server.

Table 2 changeLog Elements/Attributes

Element/Attributes	Definition and Description of Use
nameWell	Human recognizable contextual name for a well. The value for this element MUST match the value of the element nameWell in the changed data-object. If the changed data-object is well, then this value will not be defined. This element is required for objects that are a direct or indirect child of a well.
nameWellbore	Human recognizable contextual name for a wellbore. This MUST match the value of element nameWellbore in the changed object. If the changed object is well or wellbore, then this value will not be defined. This element is required for objects that are a direct or indirect child of a wellbore.
nameObject	Human recognizable contextual name for the data-object. The value for this element MUST match the value of the name element in the changed data-object.
objectType	The schema name of the (singular) data-object that changed. For example, "trajectory".
commonData	A container element that contains elements that are common to all data-objects.
customData	A container element that can contain custom or user-defined data elements.

Table 2 changeLog Elements/Attributes	
Element/Attributes	**Definition and Description of Use**
uidWell	Unique identifier for the well. The value for this attribute MUST match the value of the uidWell attribute in the changed data-object. If the changed data-object is well, then this value will not be defined. This attribute is required for data-objects that are a direct or indirect child of a well.
uidWellbore	Unique identifier for the wellbore. The value for this attribute MUST match the value of the uidWellbore attribute in the changed data-object. If the changed data-object is well or wellbore, then this value will not be defined. This attribute is required for data-objects that are a direct or indirect child of a well or wellbore.
uidObject	Unique identifier for the changed data-object. The value for this attribute MUST match the value of the uid attribute in the changed data-object.
uid	Unique identifier for the changeLog object.
lastChangeType	The type of change that occurred most recently. The value of this element MUST match the changeType element of the last entry that was added to the changeHistory.
lastChangeInfo	A description of the change that occurred most recently. The value of this element MUST match the changeInfo element of the last entry that was added to the changeHistory
changeHistory	Defines detail changes of a data-object instance. (All items below are part of the changeHistory element.)
dTimChange	The date and time when the data-object was changed. The value of this element MUST match the value of dTimLastChange in the corresponding data-object, when the changeHistory entry is added
changeType	The type of change that occurred. Valid changeType values include: add, delete, update.
changeInfo	A description of the changes.
startIndex	For non-dateTime indexes, contains the startpoint for the range that has been modified. This is the index within a growing data-object.
endIndex	For non-dateTime indexes, contains the endpoint for the range that has been modified. This is the index within a growing data-object.
startDateTimeIndex	For dateTime indexes, contains the startpoint for the range that has been modified. This is the index within a growing data-object.
endDateTimeIndex	For dateTime indexes, contains the endpoint for the range that has been modified. This is the index within a growing data-object.
mnemonics	A comma separated list with log curve mnemonics that have been changed.
sourceName	An identifier to indicate the data originator. The value of this element MUST match the value of commonData element sourceName in the data-object.
objectGrowingState	Indicates that the value of the objectGrowing flag changed and what it change to (“true” or “false”). Determines requirements for how the changeLog is updated.
updatedHeader	(Boolean) Indicates that the header was updated (through add a curve, delete a curve, change of value in header, etc.) Set to “true” if anything in header—except for objectGrowing flag—changes.

9.3.4.2 Creation and Deletion of changeLog Objects

- Each time an instance of a WITSML data-object is created in the server, the server MUST create an instance of a changeLog object. This changeLog object instance captures key change information for most changes (for exceptions, see the table below) to the corresponding WITSML data-object.

- The server MUST populate the uidObject in the changeLog schema with the uid of the data-object instance for which it was created. The uidObject is the mechanism for a changeLog instance to be associated with the data-object instance.
- After a change or deletion of a data-object, a server MUST retain the corresponding instance of the changeLog data-object for 24 hours. After 24 hours, the server MAY delete an instance of a changeLog object.
- If after a changeLog object has been deleted a change occurs in the corresponding WITSML data-object instance or the corresponding WITSML data-object instance is deleted, then a server MUST create a new instance of a changeLog object.

9.3.4.3 Update of changeLog Objects

The following table summarizes server behavior for updating a changeLog and setting change types for key client actions (add, delete and update), for the main data-object types.

		Server Update Behavior to the changeLog for Key Data-object Types			
Event	**objectGrowing state**	**Trajectory**	**Mud Log**	**Log**	**Other data-objects (non-growing)**
Add a data-object (not data item)	Any	Entry to changeLog changeType =“add”			
Delete a data-object (not a sub-node)	Any	Entry to changeLog changeType =“delete”			
Change the objectGrowing state	true → false or false →true	Entry to changeLog changeType =“update” In the changeHistory element, update the objectGrowingState with the new state (i.e., “true” or “false”)			NA
Update the data-object’s non-growing part	false	Entry to changeLog changeType =“update” In the changeHistory element, set the updatedHeader to "true”.			Entry to changeLog changeType =“update” (Note: objectGrowing state is not applicable here)
Update the data-object’s non-growing part	true	No entry to changeLog.		Entry to changeLog. changeType =“update” In the changeHistory element, set the updatedHeader to “true”.	Entry to changeLog changeType =“update” (Note: objectGrowing state is not applicable here)
Update the data-object’s growing part (including adding, modifying or deleting sub-nodes)	false	Entry to changeLog. changeType =“update” In the changeHistory element, include start/end index.		Entry to changeLog. changeType =“update” In the changeHistory element, include start/end index and mnemonics.	NA
Update the data-object’s growing part (including adding, modifying or deleting sub-nodes)	true	No entry to changeLog	Entry to changeLog only when a geology interval is deleted. For all other actions, no entry to changeLog. changeType = “update” In the changeHistory element, include start/end depth of deleted interval.	Entry to changeLog, except for append. changeType = “update” In the changeHistory element, include start/end index and mnemonics.	NA

The general rules (which are reflected in the table above) are as follows:

- For most changes to a data-object instance, the server MUST update the corresponding changeLog object instance.
- However, when objectGrowing = “true” (that is, data is being appended to a growing data-object), the server applies these special rules for updates to the changeLog object:
  - For trajectories: No updates to the changeLog objects are made.
  - For mudLogs: The server only updates the changeLog object if a geologyInterval is deleted.
  - For logs: the server updates the changeLog object if the change is not an append (i.e., change historical data, add a curve, remove a curve, change the curve’s information and so forth).
- **Special handling for the wellbore object:** The server does NOT update the changeLog for changes to these parameters in the wellbore object: md, tvd, mdBit and tvdBit.

9.3.4.4 Specifics of Updating the changeLog

A server MUST update changeLog objects in the following way:

- The server MUST add a changeHistory element to a changeLog object instance for every change in the corresponding data-object instance (with the exceptions stated above). However, the server MUST only add a single changeHistory element per WITSML function call.
- A server MUST add a changeHistory element to a changeLog element when adding, deleting, or updating data-objects. The server MUST set the changeType in the changeHistory element according to the following behavior:
  - If the changeHistory element is created as a result of the creation of a new WITSML data-object in the server, then the server MUST set the changeType to “add”.
  - If any part of the corresponding WITSML data-object changes, including: changing element values, adding or deleting elements, whether in the header or in the growing data, then the server MUST set the changeType to “update”.
  - If a main data-object is deleted. The server MUST set changeType to “delete”.
- A server MAY remove any changeHistory element from a changeLog object 24 hours after the changeHistory element was added.
- The server MUST set the element lastChangeType in the changeLog object to the value of the changeType element in the last changeHistory element that was added.
- For a growing data-object’s changeLog:
  - For any changeHistory entry that is the result of changing the growing part of the data-object, a server MUST set the appropriate index range.
  - For any changeHistory entry that is the result of changing the objectGrowing state of the data-object, a server MUST set the growingObjectState value to the new state of the objectGrowing flag (that is, “true” or “false”).
  - For any changeHistory entry that is the result of updating the data-object’s non-growing part—with the exception of the objectGrowing flag—a server MUST set the updatedHeader flag to “true”.
  - For changes to the statistical data in a growing data-object (e.g., the start and end indexes), a server MUST NOT make entries in a changeLog object.
- The value of dTimChange in a changeHistory element MUST match the value of dTimLastChange in the corresponding data-object when the changeHistory entry is added. Subsequent changes to dTimLastChange in the corresponding data-object MUST NOT affect existing values of dTimChange in the changeLog object.

9.4 Guidelines for Client Use of Change Elements

- For standard queries for the changeLog, see Section 6.6.5.17, page 84.
- When objectGrowing is “true”, a client can monitor the growing data-object for appendages by polling for any new growth since the previously known index or using the dTimLastChange of the sub-nodes.
- **Important!** A client MUST not [**else error -485**] poll too fast. The client SHOULD adapt its polling rate to the historical append rate for that data-object.

10. Custom Data

All the WITSML data-objects include a container-element named <customData> which can be used to extend the WITSML data-object definitions. While this is a useful feature, it can lead to the very interoperability problems that WITSML was intended to prevent.

For the special case of <customData>, if only the customData element is specified, then a server returns all sub-nodes. Querying sub-nodes of customData is server dependent.

To minimize problems across different implementations of custom data, observe these guidelines.

10.1 Namespace

Both clients and servers that use custom data definitions MUST use a namespace URI different from the one used by WITSML (http://www.witsml.org/...).

Example: Here is a WITSML well data-object that uses the xmlns attribute to declare a custom namespace prefix of "cns" associated with the URI http://www.my.org/schemas/100.

The data-object includes two custom data elements (cd1 and cd2) defined in that custom data namespace.

```
<?xml version="1.0" encoding="UTF-8"?>
<wells xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:cns="http://www.my.org/schemas/100">          ← custom ns declaration
       <well uid="W-1">
              … WITSML-defined elements and attributes …
              <customData>
                     <cns:cd1>value of cd1</cns:cd1>
                     <cns:cd2>value of cd2</cns:cd2>
              </customData>
       </well>
</wells>
```

Note that the WITSML-defined elements and attributes do not need a namespace prefix, because they use the default namespace declaration.

11. APPENDIX A: Defined Return Values

The values returned in the integer return value by the WITSML API functions are shown below, along with the string returned by the WMLS_GetBaseMsg function.

The WITSML-defined Return Values are in the range of -999 thru -1 (to indicate exception conditions) and positive values (to indicate no exception occurred).

Implementers are encouraged to use values from this list whenever possible.

It is recognized that this list is incomplete and additional values will need to be defined over time.

If no Return Value is defined for an error condition, implementers are free to assign their own proprietary Return Values less than -1000 (i.e., -1001, -1002, etc.).

It is hoped that proprietary Return Values will be contributed to the WITSML community for possible inclusion in this standard list.

Value	String returned by WMLS_GetBaseMsg
1	Function completed successfully
2	Partial success: Function completed successfully but some growing data-object data-nodes were not returned.
-401	The input template MUST contain a plural root element.
-402	The value of the OptionsIn keyword of 'maxReturnNodes' MUST be greater than zero.
-403	A template must include a default namespace declaration for the WITSML namespace.
-404	In the value of the capClient schemaVersion, the oldest Data Schema Version must be listed first, followed by the next oldest, etc.
-405	For WMLS_AddToStore, a data-object with the same type and unique identifier(s) must NOT already exist in the persistent store.
-406	For WMLS_AddToStore, all parentage-pointers and lower level (child) uid values must be defined in the XMLin file.
-407	A non-empty value must be defined for WMLtypeIn.
-408	A non-empty value must be defined for the input template.
-409	The input template must conform to the appropriate derived schema.
-410	A template containing empty values must otherwise conform to the appropriate derived schema.
-411	The OptionsIn parameter string must be encoded utilizing a subset (semicolon separators and no whitespace) of the encoding rules for HTML form content type application/x-www-form-urlencoded.
-412	The result of an add, update, or delete operation must be compliant with the derived write schema after performing a retrieval of the complete data-object.
-413	The data-object must be supported by the server as defined in its capability data-object.
-414	In the server configuration for the site, the user must have rights to perform the requested operation on the data-object.
-415	The input template must specify the unique identifiers of one data-object to be processed.
-416	For WMLS_DeleteFromStore, an empty uid attribute must not be specified.
-417	For WMLS_DeleteFromStore, an empty uom attribute must not be specified.
-418	For WMLS_DeleteFromStore, if an element with a uid attribute in the schema is specified then it must also be specified with a value for its uid attribute.
-419	For WMLS_DeleteFromStore, an empty non-recurring container-element with no unique identifier in the schema or an empty value for logData must not be specified.

Value	String returned by WMLS_GetBaseMsg
-420	For WMLS_DeleteFromStore, an empty node must not be specified for a non-recurring element or attribute that is mandatory in the write schema.
-421	For WMLS_DeleteFromStore, a recurring element that is mandatory in the write schema must retain at least one occurrence after the deletion.
-422	For WMLS_GetBaseMsg, a non-empty value must be specified for ReturnValueIn.
-423	For WMLS_GetCap, the OptionsIn keyword 'dataVersion' must specify a Data Schema Version that is supported by the server as defined by WMLS_GetVersion.
-424	For WMLS_GetCap, the OptionsIn keyword 'dataVersion' must be specified.
-425	For WMLS_GetFromStore, the OptionsIn keyword 'returnElements' must not specify a value of "header-only" or "data-only" for a non-growing data-object.
-426	The input template must conform to the appropriate derived schema after uncompressing the string.
-427	The OptionsIn keyword 'requestObjectSelectionCapability' with a value other than 'none' must not be specified with any other OptionsIn keyword.
-428	If the OptionsIn keyword 'requestObjectSelectionCapability' is specified with a value other than 'none' then QueryIn must be the Minimum Query Template.
-429	For WMLS_GetFromStore, the logData section must not recur when retrieving data.
-430	A client must specify an item for data-item selection that the server supports.
-431	A client must specify an item (element or attribute) for data-object selection that the server supports.
-432	If cascading deletes are not invoked, a client must only request deletion of bottom level data-objects such that all child data-objects are deleted before the parent is deleted.
-433	A data-object with the same type and unique identifier(s) must already exist in the persistent store.
-434	For updating systematically growing data, if data-nodes are specified then the column-identifiers (mnemonic) must be specified in the data-column-list.
-435	When an update adds a new column-identifier to a systematically growing data-object, an old (existing) column-identifier (other than the index-column) must not be specified with the new column-identifier.
-436	When an update adds a new column-identifier to a systematically growing data-object, a structural-range must not also be specified.
-437	For WMLS_DeleteFromStore with a systematically growing data-object, a column-identifier must not be specified in the data-column-list (mnemonicList).
-438	When multiple selection criteria is are included in a recurring element, the same selection items must exist in each recurring node.
-439	When multiple selection criteria is are included in a recurring pattern, an empty value must not be specified.
-440	The OptionsIn value must be a recognized keyword for that function.
-441	The value specified with an OptionsIn keyword must be a recognized value for that keyword.
-442	A client must not specify an OptionsIn keyword that is not supported by the server.
-443	The value of the uom attribute is must match an 'annotation' attribute from the WITSML Units Dictionary XML file.
-444	The input template must not specify more than one data-object.
-445	For WMLS_UpdateInStore, new elements or attributes must not be empty.
-446	For WMLS_UpdateInStore, a uom attribute must not be specified unless its corresponding value is specified.
-447	When adding or updating curves, column-identifiers must be unique.

Value	String returned by WMLS_GetBaseMsg
-448	For WMLS_UpdateInStore, if an element with a unique identifier in the schema is specified then the identifier value must also be specified
-449	When updating data in a systematically growing data-object the indexCurve must be specified in the mnemonicList.
-450	When updating data in a systematically growing data-object, each mnemonic must occur only once in the mnemonicList.
-451	When updating data in a systematically growing data-object, the unitList must always be specified.
-452	When updating data in a systematically growing data-object, the unitList must be populated with units that match the header.
-453	For WMLS_AddToStore and WMLS_UpdateInStore, the client must always specify the unit for all measure data.
-454	For a particular WMLS_AddToStore call, a client must specify all growing data-object index data in the same unit of measure.
-455	All datum elements or attributes for indexes in a growing data-object must implicitly or explicitly point to the same wellDatum when adding or updating data.
-456	The client must not attempt to add or update more data than is allowed by the server's maxDataNodes and maxDataPoints values.
-457	If a column-identifier representing the index column is specified then it must be specified first in the data-column-list.
-458	For growing data-objects, a combination of depth and date-time structural-range indices must not be specified.
-459	For systematically growing data-objects, the column-identifier (mnemonic) values must not contain one of the following special characters: single-quote, double-quote, less than, greater than, forward slash, backward slash, ampersand, comma.
-460	For getting systematically growing data-objects, if column-identifiers (mnemonics) are specified in both the header and data sections then the column-identifiers must be the same in the two sections.
-461	For getting systematically growing data-objects, if a column-definition (logCurveInfo) section is specified then a mnemonic element must be specified.
-462	For getting systematically growing data-objects, if a data-node (logData) section is specified then a mnemonicList element must be specified.
-463	For updating systematically growing data, the update data must not contain multiple nodes with the same index.
-464	Each lower level child uid value must be unique within the context of its nearest recurring parent node.
-465	The capClient apiVers value must match the API schema version.
-466	The CapabilitiesIn XML MUST conform to the API capClient schema.
-467	The server does not support the API Version provided by the client.
-468	A QueryIn template must include a version attribute in the plural data-object that defines the Data Schema Version of the data-object.
-469	The query template is does not conform to the data schema for this data-object.
-470	<Not Assigned>
-471	<Not Assigned>
-472	The client product name and/or product version number are missing in the HTTP user-agent field. WITSML requires this information for each function call.
-473	For the capClient object, the values of schemaVersion must match the version attribute used in the plural data-objects.

Value	String returned by WMLS_GetBaseMsg
-474	For growing data-objects in a query, if a node-index value is to be returned, then the client must explicitly specify it in the query (which is true for any element or attribute in a query).
-475	No subset of a growing data-object is specified. A query must specify a subset of one growing data-object per query, but multiple individual queries may be included inside the query plural data-object.
-476	For WMLS_GetFromStore: OptionsIn keyword *returnsElements=latest-change-only* can only be used for a changeLog object.
-477	For getting systematically growing data-objects, both column-description and data sections must be specified.
-478	Incorrect case on parent uid.
-479	Unable to decompress query.
-480	A new column cannot be added at the same time an existing column is being updated. (The index column does not count as being updated.)
-481	Parent does not exist.
-482	Duplicate mnemonics in a query are not allowed.
-483	The XMLin document does not comply with the update schema.
-484	A mandatory write schema item is missing.
-485	The client is polling too fast.
-486	In WMLS_AddToStore, the WMLtypeIn objectType must match the XMLin objectType. Currently, they do not match.
-487	In WMLS_AddToStore, the objectType being added in WMLtypeIn must be an objectType supported by the server. The server does not support the object type trying to be added.

12. APPENDIX B: Server Object Selection Capability Templates

The values returned are RECOMMENDED to conform to the following based on the underlying type of the value.

Data Type	Value
Boolean	false or 0 **NOTE**: The XML standard uses both “false” and “0” or “true” and “1” for Boolean values.
string	abc
date	1900-01-01
dateTime	1900-01-01T00:00:00.000Z
timeZone	Z
number	1
Enumerated string	A valid enumeration value.
calendarYear	1000
iadcBearingWearCode	E
geodeticZoneString	60N
sectionNumber	36
publicLandSurveySystemQuarterTownship	NE
publicLandSurveySystemQuarterSection	NE

13. APPENDIX C: WITSML Data-objects

WITSML v1.4.1 has data schemas for the data-objects listed here.

Data-object	Purpose
attachment	Provides a central location for finding a digital attachment that is associated with another WITSML well related data-object. The attachment is captured in a base 64 binary type.
bhaRun	Captures information about one run of the drillstring into and out of the hole. The drillstring configuration is described in the tubular data-object. That is, one configuration may be used for many runs. This data-object is uniquely identified within the context of one wellbore data-object.
cementJob	Captures information about cementing operations which are undertaken to seal the annulus after a casing string has been run, to seal a lost circulation zone or to set a plug to support directional drilling operations or seal a well so that it may be abandoned. This data-object is uniquely identified within the context of one wellbore data-object.
changeLog	Provides a central location for knowledge of changes in the other data-objects. This data-object only makes sense within the context of a WITSML server because there is no domain data in the data-object.
convCore	Captures information about a conventional core. This includes a description of the geology and lithology over the interval. See also sidewallCore. This data-object is uniquely identified within the context of one wellbore data-object.
coordinateReferenceSystem	Defines the details about a Coordinate Reference System (CRS) using GML version 3.2.1 schemas as defined at http://schemas.opengis.net/gml/3.2.1/. A best practice would be to reference a well known CRS as defined by the EPSG (http://www.epsg.org/) rather than using the coordinateReferenceSystem data-object to define one from scratch.
drillReport	Captures a daily drilling report focused on reporting from the operator to partners or to a governmental agency. See the opsReport data-object for a similar report whose focus is service company to operator. This data-object is uniquely identified within the context of one wellbore data-object.
fluidsReport	Captures an analysis of the drilling mud. This data-object is uniquely identified within the context of one wellbore data-object.
formationMarker	Captures information about a geologic formation that was encountered in a wellbore. This data-object is uniquely identified within the context of one wellbore data-object.
log	It was designed primarily for MWD logs and is constrained to capturing curves from a single pass. This data-object is uniquely identified within the context of one wellbore data-object. For multi-pass log capabilities (previously covered by the wellLog data-object), see *objectGroup* below.

Data-object	Purpose
message	Provides a central location for informative time stamped information about another well related data-object. These messages can represent alarms, warnings, events, etc.
mudLog	Captures information in a mud log. This includes lithologic information derived from cuttings encountered on a shaker screen at a site. This data-object is uniquely identified within the context of one wellbore data-object.
objectGroup	New data-object added in v 1.4.1.0. Primary purpose is to provide a way to logically group logs that are part of a multi-pass/multi-run suite of logs or tests—logs that a user would only want to see as part of a collection. It has been designed to be used to group any WITSML data-objects, if use cases determine the necessity.
opsReport	Captures a daily drilling report focused on reporting from the service company to the operator. See the drillReport for a similar data-object whose focus is operator to partner or to governmental agency. This data-object is uniquely identified within the context of one wellbore data-object.
rig	Captures information about a drilling rig used to drill a wellbore. This data-object is uniquely identified within the context of one wellbore data-object.
risk	Provides a central location for capturing risk information about other well related data-objects.
sidewallCore	Captures information about a core from the side of a borehole. See also convCore. This data-object is uniquely identified within the context of one wellbore data-object.
stimJob	Captures a post job summary report about a stimulation (fracture) job. See the WITSML stimJob data-object Usage Guide for an overview of its intended usage. This data-object is uniquely identified within the context of one wellbore data-object.
surveyProgram	Captures information about the nature, range and sequence of directional surveying tools run in a wellbore for the management of positional uncertainty. This data-object is uniquely identified within the context of one wellbore data-object.
target	Captures information about intended targets of a trajectory survey. This data-object is uniquely identified within the context of one wellbore data-object.
toolErrorModel	Defines a surveying tool error model. This data-object is globally unique.
toolErrorTermSet	Defines a set of surveying tool error terms which may be used in a toolErrorModel. This data-object is globally unique.
trajectory	Captures information about a directional survey in a wellbore. It contains many trajectory stations to capture the information about individual survey points. This data-object is uniquely identified within the context of one wellbore data-object.
tubular	Captures information about the configuration of a drill string. See the bhaRun data-object for information about a use of this configuration. This data-object is uniquely identified within the context of one wellbore data-object.

Data-object	Purpose
wbGeometry	Captures information about the configuration of the permanently installed components in a wellbore. It does not define the transient drilling strings (see the tubular data-object) or the hanging production components. This data-object is uniquely identified within the context of one wellbore data-object.
well	Captures the general information about a well. This might sometimes be called a well header. All information which is the same for all wellbores (sidetracks) is contained in the well data-object.
wellbore	Captures the general information about a wellbore. This might sometimes be called a wellbore header. A wellbore represents the path from surface to a unique bottomhole. The wellbore data-object is uniquely identified within the context of one well data-object.

14. APPENDIX D: Examples

This appendix contains some example WITSML workflows and many examples for growing data-objects.

14.1 Example Workflows

This section contains some high-level example workflows.

14.1.1 Real-time Sensor Information

This example shows how real-time sensor information can be transferred using the STORE interface from a sensor aggregator application on a rig to an application in the operating company's offices. Although, if a WITSML server existed at the wellsite, that would be the preferred target application.

The sensor aggregator system is the client and the office system is the server. The real-time information is "pushed" from the client to the server.

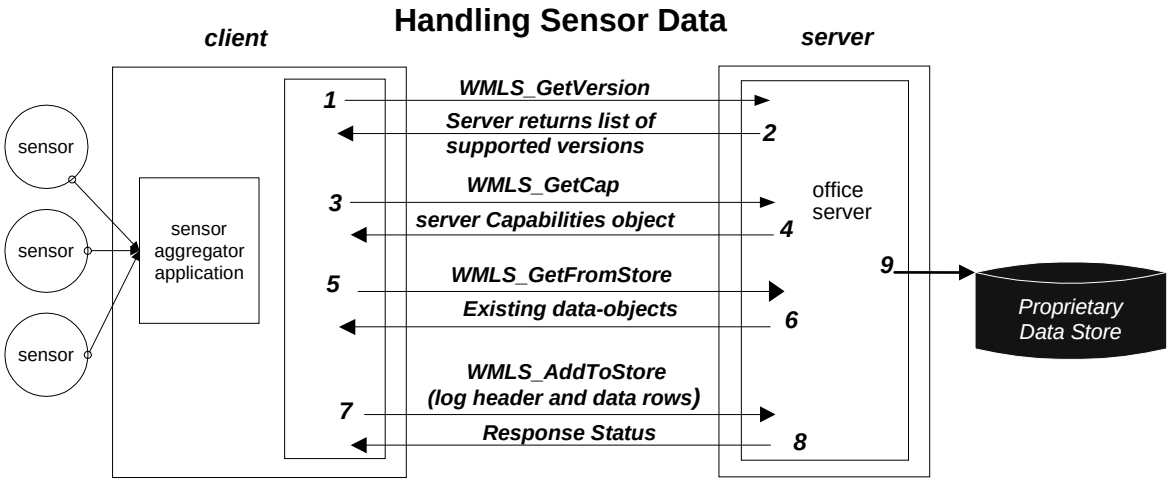


Figure 4 Sensor Data Scenario

The step numbers below refer to numbers in Figure 4.

1. First, the client needs to know which version(s) of WITSML the server supports; it does this using WMLS_GetVersion function (see page 44).
2. The server returns a list of the versions that it supports.
3. Next the client needs to know the capabilities of the server, that is, what data-object types and functions it supports. The client does this using the WMLS_GetCap (Get Capabilities) function (see page 45).
4. The server's WMLS_GetCap function returns the server's Capabilities Object to the client, which lists the data-object types the server can support for each function. In this example, for the client to proceed, the server only needs to list the log data-object type with the WMLS_AddToStore function (see page 47).
5. The client then needs to know what parent data-objects are available for the log data-object, so it queries the server for this information using the WMLS_GetFromStore function (see page 54). Because parent data-object information can vary over time, this information cannot be stored in the server's capabilities data-object, thus the need to query for it.

The following example query returns the identifiers for all well data-objects on the server.

```
<wells xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
   <well uid="">
         <name />
   </well>
</wells>
```

Depending on client capabilities, it may also know the well/wellbore identifiers. If it does not know the wellbores, then it needs to determine the identifiers of the wellbores associated with that well using the "uid" value returned from well query, such as the following:

```
<wellbores xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
   <wellbore uid="" uidWell="uid-returned-by-the-well-query">
         <name />
   </wellbore>
</wellbores>
```

"uid-returned-by-the-well-query" is replaced by the actual uid for one well that returned from the query for all the wells.

6. The server returns to the client the identifiers of the well and wellbore(s).
7. Using the WMLS_AddToStore function (see page 44), the client then adds a new log to the server. The new log will normally contain the header that describes the curves and may also define zero or more data rows. This is specified in a template similar to the following, which specifies only two curves: the index curve "MDepth" and the data curve "TQ on btm".

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
   <log uidWell="uid-returned-by-the-well-query"
        uidWellbore="uid-returned-by-the-wellbore-query" uid="uid-return-by-
WMLS_AddToStore">
         <nameWell>6507/7-A-42</nameWell>
         <nameWellbore>A-42</nameWellbore>
         <name>new log</name>
         <serviceCompany>My Company</serviceCompany>
         <indexType>measured depth</indexType>
         <direction>increasing</direction>
         <indexCurve>Mdepth</indexCurve>
         <logCurveInfo uid="Mdepth">
               <mnemonic>Mdepth</mnemonic>
               <classWitsml>measured depth of hole</classWitsml>
               <unit>m</unit>
               <typeLogData>double</typeLogData>
         </logCurveInfo>
         <logCurveInfo uid="TQ_on_btm">
               <mnemonic>TQ on btm</mnemonic>
               <classWitsml>torque (average)</classWitsml>
               <unit>kft.lbf</unit>
               <curveDescription>On bottom torque</curveDescription>
               <traceState>raw</traceState>
               <typeLogData>double</typeLogData>
         </logCurveInfo>
         <logData>
               <mnemonicList>Mdepth,TQ on btm</mnemonicList>
               <unitList>m,kft.lbf</unitList>
               <data>498,-0.33,0.1</data>
         </logData>
   </log>
</logs>
```

8. Because a uid for the log was not specified, WMLS_AddToStore assigns a unique value and returns it to the client. If a uid value had been specified for the log, then the server would have honored it, unless it was not unique within the context of the wellbore. But, to determine what uid values had already been used, the client would need to query the server for the identity of all logs that are dependent on that wellbore.

   Now the client is ready to start sending additional data values to the server by repetitively invoking WMLS_AddToStore (as in step 7) using a template such as the following to add one or more rows to the log.

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
   <log uidWell="uid-returned-by-the-well-query"
        uidWellbore="uid-returned-by-the-well-query"
        uid="uid-returned-by-WMLS_AddToStore">
          <logData>
                <mnemonicList>Mdepth,TQ on btm</mnemonicList>
                <unitList>m,kft.lbf</unitList>
                <data>499,1.25,0</data>
                <data>500.01,1.42</data>
          </logData>
   </log>
</logs>
```

9. The server persists (stores) the data on its internal data store.

14.1.2 Rig Site Repository/Aggregator

This example shows how a server located at a rig site can act as a central repository for data related to its well(s).

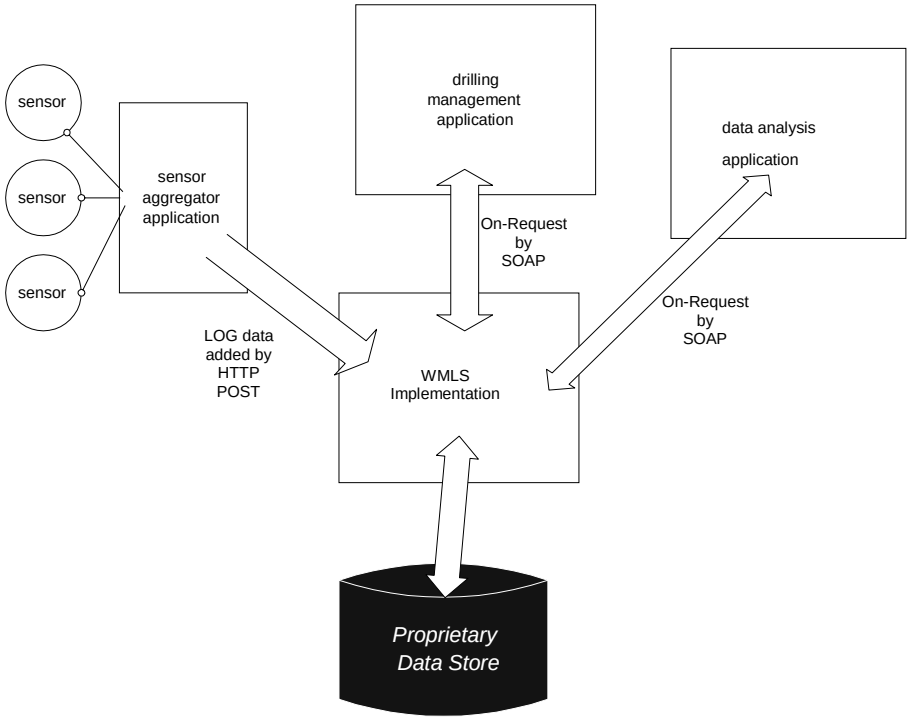


Figure 5 Aggregator Scenario

Some of the data stored in such a repository could be collected by other front-end applications used on the rig, such as drilling management systems. These applications would act as clients, and use the rig server as a persistent store, to either augment or replace their own internal proprietary databases. These applications could provide the user interface for initially defining the well, accepting manual input, and for producing and distributing periodic reports.

Other applications on the rig could use the server as a realtime data repository. Sensor aggregator applications would collect realtime data and add it to a log data-object on the rig server.

Backend applications located either at the rig or at the office could then act as clients and retrieve data from the server on request. A data analysis application, for example, could populate its own well definition based on the well data-object it retrieves from the server, and then retrieve the stored log curves for analysis.

Most of the internal processes involved in receiving the realtime data and persisting it are the same as for the sensor use case described above; most of the details have been omitted here.

To retrieve a WITSML data-object stored on a server, the client application passes the type of WITSML data-object and a Query template as parameters to the WMLS_GetFromStore function, as shown in the following example code:

```
   RetVal = WMLS_GetFromStore("well",                ← data-object type
         "<wells…><well uid="12""><name/></well></wells>",← Query template
         OptionsIn,
         CapabilitiesIn,
         XMLout,                                      ← returned data-objects
         SuppMsgOut)
```

The server's STORE implementation returns an XML document containing the WITSML data-object(s) that satisfy the Query template, and a return value (RetVal) indicating the success or failure of the function.

The client application can also add a single WITSML data-object to a server by passing an XML document containing the data-object to the WMLS_AddToStore function:

```
RetVal = WMLS_AddToStore("well",
                        StringContainingXML,
                        OptionsIn,
                        CapabilitiesIn,
                        SuppMsgOut)
```

And likewise, the client application can update or delete a single WITSML data-object stored on the server using the WMLS_UpdateInStore and WMLS_DeleteFromStore functions.

14.2 Growing Object Examples

The special handling of growing objects, which include trajectory, mudLog and log data-objects, allows a query template to specify a range of recurring structures to be returned.

For more information on growing objects, see

14.2.1 How to Specify Recurring Structures to be Returned

For all three data-objects, the range of recurring structures to be returned is indicated by specifying values for specific, pre-determined index range data element(s) in the non-recurring ("header") portion of the data-object. These values are then compared against specific, pre-determined index data element(s) in the recurring portion of the data-object. If the data element(s) in the recurring portion fall within the specified index range, that occurrence of the recurring structure is included in the returned data-object.

Object	Range Specified By Element(s) in Header	Recurring Structure	Compared To Field in Recurring Structure
example (see below)	begin/end	xxx	index
mudLog	startMd/endMd	geologyInterval	mdTop/mdBottom
log	startIndex/endIndex *	data **	data value of the index
trajectory	mdMn/mdMx	trajectoryStation	md

* startIndex and endIndex elements are in the header. The elements startDateTimeIndex and endDateTimeIndex are alternative indexes which are treated similarly to startIndex and endIndex but have a different underlying type.

**data structure is in the logData container

Example		
Persisted Data-object	**Query Template**	**Returned Data-object**
<example…> <header_elements> … <xxx> <index>500</index> … </xxx> <xxx> <index>1000</index> … </xxx> <xxx> <index>3000</index> … </xxx> </example>	<example…> <begin>950</begin> <end>4000</end> <xxx> <index/> </xxx> </example…>	<example…> <begin>1000</begin> <end>3000</end> <xxx> <index>1000</index> </xxx> <xxx> <index>3000</index> </xxx> </example>

14.2.2 Trajectory Data-object Example

The trajectory data-object contains recurring trajectoryStation structures:

```
<trajectorys xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
      <trajectory uidWell="W-12" uidWellbore="B-01" uid="pe84e">
            <trajectoryStation uid="34ht5">
                  …
            </trajectoryStation>
            <trajectoryStation uid="34ht6">
                  …
            </trajectoryStation>
      </trajectory>
</trajectorys>
```

Trajectory data-object with multiple trajectoryStation structures.

A query can retrieve a range of trajectoryStations by specifying values for the minimum and maximum measured-depth elements (mdMn and mdMx) of the trajectory data-object and by specifying a single trajectoryStation element in the query template, such as:

```
<trajectorys xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
      <trajectory uidWell="W-12" uidWellbore="B-01" uid="pe84e">
            <mdMn uom="ft">3000</mdMn>
            <mdMx uom="ft">4000</mdMx>
                  …
            <trajectoryStation uid="">
                  …
            </trajectoryStation>
      </trajectory>
</trajectorys>
```

Query template specifying a depth range and a single trajectoryStation structure.

14.2.2.1 Special Handling Summary

The values for **mdMn** and **mdMx** are interpreted as meaning to include in the returned trajectory data-object(s) all instances of the trajectoryStation structure where the measured depth element (**md**) of the trajectoryStation structure is greater than or equal to **mdMn** and less than or equal to **mdMx**.

- If no value is specified for both the mdMn or mdMx elements, then standard STORE Interface query template handling rules apply, and all trajectory stations—regardless of depth—are returned (assuming they match any other specified selection criteria).
- If a value is specified for mdMn and not for mdMx, then all trajectoryStation occurrences with an md value greater than or equal to mdMn will be returned. Likewise, if a value is specified for mdMx and not for mdMn, all trajectoryStation occurrences with an md value less than or equal to mdMx will be returned.
- If there are no trajectoryStations with md values within the specified range of value(s) for mdMn/mdMx, then nothing is returned because not all criteria have been met

As is true for STORE query templates in general, if the md element of the trajectoryStation structure is to be returned, then it must be explicitly specified in the query:

```
<trajectorys xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
      <trajectory uidWell="W-12" uidWellbore="B-01" uid="pe84e">
            <mdMn uom="ft">3000</mdMn>
            <mdMx uom="ft">4000</mdMx>
                  …
            <trajectoryStation>
                  <md uom=""/>
            </trajectoryStation>
      </trajectory>
</trajectorys>
```

Query template requesting that the md data item be returned in the trajectoryStation structure(s).

The selection of which trajectoryStation(s) are to be returned is performed against the persisted md value(s), whether the md item is specified in the query template or not.

If a value is specified in the trajectory for mdMn or mdMx, and a value is also specified in the trajectoryStation for md, then the value of md in the trajectoryStation is ignored:

```
<trajectorys xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
      <trajectory uidWell="W-12" uidWellbore="B-01" uid="pe84e">
            <mdMn uom="ft">3000</mdMn>
            <mdMx uom="ft">4000</mdMx>
                  …
            <trajectoryStation>
                  <md uom="ft">5000</md>
            </trajectoryStation>
      </trajectory>
</trajectorys>
```

This value will be ignored.

The above query returns all trajectoryStation instances between 3000 and 4000 (inclusive). The md element of the returned trajectoryStation contains the persisted value for each.

Any values specified for an attribute or element of the trajectoryStation structure—other than the md child element—are combined with the selection criteria specified by mdMn and mdMx.

The following example returns each trajectoryStation within the 3000-4000 depth range that have changed since the specified time:

```
<trajectorys xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
      <trajectory uidWell="W-12" uidWellbore="B-01" uid="pe84e">
            <mdMn uom="ft">3000</mdMn>
            <mdMx uom="ft">4000</mdMx>
                  …
            <trajectoryStation>
                  <commonData>
                        <dTimLastChange>2010-12-31T12:02:36.300Z</dTimLastChange>
                  </commonData>
            </trajectoryStation>
      </trajectory>
</trajectorys>
```

Criteria are combined.

14.2.3 mudLog Data-object Example

The mudLog data-object contains recurring parameter and geologyInterval structures:

```
<mudLogs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
      <mudLog uidWell="W-12" uidWellbore="B-01" uid="h45a">
            …
            <parameter …>
                  …
            </parameter>
            <parameter …>
                  …
            </parameter>
            <geologyInterval …>
                  …
            </geologyInterval>
            <geologyInterval …>
                  …
            </geologyInterval>
            …
      </mudLog>
</mudLogs>
```

mudLog data-object with multiple parameter and geologyInterval structures

A query template can retrieve a range of parameters and/or geologyIntervals by specifying values for the start and end measured-depth elements (startMd and endMd) of the mudLog data-object:

```
<mudLogs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
      <mudLog uidWell="W-12" uidWellbore="B-01" uid="h45a">
            <startMd uom="ft">5010</startMd>
            <endMd uom="ft">5020</endMd>
            <geologyInterval …>
                  …
            </geologyInterval>
      </mudLog>
</mudLogs>
```

Query template specifying a depth range and a single geologyInterval structure.

14.2.3.1 Special Handling Summary

The values for startMd and endMd are interpreted as meaning to include in the returned mudLog data-object(s) all instances of the parameter and geologyInterval structures where the following relationship (controlled by the OptionsIn parameter "intervalRangeInclusion" explained in Section 6.6.4.3, page 61) exists between:

- the measured depth top element (mdTop) of the parameter or geologyInterval and the startMd of the mudLog query
- The measured depth bottom element (mdBottom) of the parameter or geologyInterval and the endMd of the mudLog query.

If you use this value for intervalRangeInclusion:	The Query Returns Only:
whole-interval	Those parameter/geologyIntervals whose mdTop is greater than or equal to the requested startMd –AND– whose mdBottom is less than or equal to than the requested endMD.
minimum-point (DEFAULT VALUE)	Those parameter/geologyIntervals whose mdTop is greater than or equal to the requested startMd and is less than or equal to the requested endMD. (The mdBottom value of parameter/geologyInterval is ignored for the comparison.)
any-part	Those parameter/geologyIntervals whose mdTop is less than or equal to the requested endMd –AND– whose mdBottom is greater than or equal to than the requested startMD.

- If no value is specified for both the startMd or endMd elements, standard STORE Interface query template handling rules apply, and all parameter or geologyIntervals—regardless of depth—are returned (assuming they match any other specified selection criteria).
- If a value is specified for startMd and not for endMd, then all parameter or geologyInterval occurrences with an mdTop value greater than or equal to startMd are returned. Likewise, if a value is specified for endMd and not for startMd, then all parameter or geologyInterval occurrences with an mdBottom value less than or equal to endMd are returned.
- If there are no parameters or geologyIntervals with mdTop/Bottom values within the specified range of value(s) for startMd/endMd, then nothing is returned because not all criteria will have been met.

As is true for STORE query templates in general, if the mdTop and/or mdBottom elements of the parameter or geologyInterval structure are to be returned, then they must be specified:

```
<mudLogs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
	<mudLog uidWell="W-12" uidWellbore="B-01" uid="h45a">
		<startMd uom="ft">5011</startMd>
		<endMd uom="ft">5019</endMd>
		<geologyInterval …>
			<mdTop uom=""/>
			<mdBottom uom=""/>
		</geologyInterval>
	</mudLog>
</mudLogs>
```

The returned data from this query is a single mudLog object, corresponding to the specified uid value. The range of geologyInterval sub-objects is restricted by the query parameters for start and end depths. In this example, the geologyInterval sub-objects are at 2-foot intervals. This means that if the requested range is 5011 to 5019 feet:

- If intervalRangeInclusion=whole-interval: a geologyInterval from 5010 to 5012 feet is not returned because its start depth is outside the requested range of 5011 to 5019 feet; similarly a geologyInterval from 5018 to 5020 feet is not returned because its end depth is greater than the end of the requested interval.
- If intervalRangeInclusion=minimum-point, a geologyInterval from 5010 to 5012 feet is not returned because its start depth is outside the requested range of 5011 to 5019 feet.
- If intervalRangeInclusion=any-part, the above geologyInterval elements from 5010 to 5012 feet and from 5018 to 5020 feet will be returned because a part of their interval overlaps the requested range of 5011 to 5019 feet.

XMLOut when intervalRangeInclusion=whole-interval:

```
<mudLogs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
	<mudLog uidWell="W-12" uidWellbore="B-01" uid="h45a">
		<startMd uom="ft">5012</startMd>
		<endMd uom="ft">5018</endMd>
		<geologyInterval uid="gi123">
			<mdTop uom="ft">5012</mdTop>
			<mdBottom uom="ft">5014</mdBottom>
		</geologyInterval>
	<	geologyInterval uid="gi124">
			<mdTop uom="ft">5014</mdTop>
			<mdBottom uom="ft">5016</mdBottom>
		</geologyInterval>
		<geologyInterval uid="gi125">
			<mdTop uom="ft">5016</mdTop>
			<mdBottom uom="ft">5018</mdBottom>
		</geologyInterval>
	</mudLog>
</mudLogs>
```

XMLOut when intervalRangeInclusion=minimum-point:

```
<mudLogs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
	<mudLog uidWell="W-12" uidWellbore="B-01" uid="h45a">
		<startMd uom="ft">5012</startMd>
		<endMd uom="ft">5020</endMd>
		<geologyInterval uid="gi123">
			<mdTop uom="ft">5012</mdTop>
			<mdBottom uom="ft">5014</mdBottom>
		</geologyInterval>
		<geologyInterval uid="gi124">
			<mdTop uom="ft">5014</mdTop>
			<mdBottom uom="ft">5016</mdBottom>
		</geologyInterval>
```

```
            <geologyInterval uid="gi125">
                  <mdTop uom="ft">5016</mdTop>
                  <mdBottom uom="ft">5018</mdBottom>
            </geologyInterval>
            <geologyInterval uid="gi126">
                  <mdTop uom="ft">5018</mdTop>
                  <mdBottom uom="ft">5020</mdBottom>
            </geologyInterval>
      </mudLog>
</mudLogs>
```

XMLOut when intervalRangeInclusion=any-part:

```
<mudLogs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
      <mudLog uidWell="W-12" uidWellbore="B-01" uid="h45a">
            <startMd uom="ft">5010</startMd>
            <endMd uom="ft">5020</endMd>
            <geologyInterval uid="gi122">
                  <mdTop uom="ft">5010</mdTop>
                  <mdBottom uom="ft">5012</mdBottom>
            </geologyInterval>
            <geologyInterval uid="gi123">
                  <mdTop uom="ft">5012</mdTop>
                  <mdBottom uom="ft">5014</mdBottom>
            </geologyInterval>
            <geologyInterval uid="gi124">
                  <mdTop uom="ft">5014</mdTop>
                  <mdBottom uom="ft">5016</mdBottom>
            </geologyInterval>
            <geologyInterval uid="gi125">
                  <mdTop uom="ft">5016</mdTop>
                  <mdBottom uom="ft">5018</mdBottom>
            </geologyInterval>
            <geologyInterval uid="gi126">
                  <mdTop uom="ft">5018</mdTop>
                  <mdBottom uom="ft">5020</mdBottom>
            </geologyInterval>
      </mudLog>
</mudLogs>
```

14.2.4 log Data-object Example

14.2.4.1 Overview of the Log data-object

Because the special handling needed by the log data-object is somewhat more complex than the other data-objects, let's first review the structure of the log data-object.

The log data-object is logically divided into two sections: metadata about the log (header section) and the data for the curves (logData section).

XML Element	Description
<log …>	
<indexType …>	The kind of index curve.
<startIndex …>	starting depth index for entire logData section
<endIndex …>	ending depth index for entire logData section
<startDateTimeIndex …>	starting time index for entire logData section
<endDateTimeIndex …>	ending time index for entire logData section
…	additional child elements of header
<logCurveInfo …>	metadata about a curve
<minIndex …>	minimum depth index for this curve
<maxIndex …>	maximum depth index for this curve
<minDateTimeIndex …>	minimum time index for this curve
<maxDateTimeIndex …>	maximum time index for this curve

XML Element	Description
…	additional child elements of logCurveInfo
</logCurveInfo>	
…	additional occurrences of logCurveInfo structure
<logData …>	data section
<mnemonicList>	List of curve mnemonics in <data>
<unitList>	List of curve units in <data>
<data>	data for all curves at this time/depth index
…	additional occurrences of data element
</logData>	
</log>	

The header section defines:

- How the log is indexed. Typically some sort of depth or a timestamp: the indexType element.
- The start and end depth/time values for the entire log: startIndex and endIndex or startDateTimeIndex and endDateTimeIndex.
- Information about the log curves: one or more logCurveInfo elements, which in turn contain:
  - minimum and maximum depth/time values for this particular curve (the minIndex and maxIndex or minDateTimeIndex and maxDateTimeIndex elements).
  - the underlying data representation
  - (optionally) the string to be used to represent a null value for this log curve
- Which curve serves as the indexing curve: indexCurve (the name of the curve that contains the time or depth value with which the other data points are associated).

The logData section of the log data-object contains:

- One or more <data> elements, each containing a comma-delimited string value that contains the data points at a particular depth or time for all the curves in the log.
- A list of curve mnemonics and list of curve units representing the list of data values in the <data> element. These lists represent a denormalized view of the equivalent information in the logCurveInfo section. The mnemonic list defines the curve order in the other lists.

14.2.4.2 Example

Here is an example of a simplified log data-object containing four log curves: Mdepth, ROP, Bit RPM and ECD.

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
      <log uidWell="W-12" uidWellbore="B-01" uid="L001">
            <nameWell>example</nameWell>
            <nameWellbore>example</nameWellbore>
            <name>example</name>
            <indexType>measured depth</indexType>
            <startIndex uom="ft">4050</startIndex>
            <endIndex uom="ft">4090</endIndex>
            <indexCurve>Mdepth</indexCurve>
            <logCurveInfo uid="mdepth">
                  <mnemonic>Mdepth</mnemonic>
                  <unit>ft</unit>
                  <minIndex uom="ft">4050</minIndex>
                  <maxIndex uom="ft">4090</maxIndex>
                  <typeLogData>double</typeLogData>
            </logCurveInfo>
            <logCurveInfo uid="rop">
                  <mnemonic>ROP</mnemonic>
                  <unit>ft/hr</unit>
                  <minIndex uom="ft">4050</minIndex>
                  <maxIndex uom="ft">4090</maxIndex>
                  <typeLogData>double</typeLogData>
            </logCurveInfo>
            <logCurveInfo uid="bit_rpm">
                  <mnemonic>Bit RPM</mnemonic>
                  <unit>rpm</unit>
                  <nullValue>-99999</nullValue>
                  <minIndex uom="ft">4050</minIndex>
                  <maxIndex uom="ft">4070</maxIndex>
                  <typeLogData>double</typeLogData>
            </logCurveInfo>
            <logCurveInfo uid="ecd">
                  <mnemonic>ECD</mnemonic>
                  <unit>g/cm3</unit>
                  <minIndex uom="ft">4060</minIndex>
                  <maxIndex uom="ft">4090</maxIndex>
                  <typeLogData>double</typeLogData>
            </logCurveInfo>
            <logData>
                  <mnemonicList>Mdepth,ROP,Bit RPM,ECD</mnemonicList>
                  <unitList>ft,ft/hr,rpm,g/cm3</unitList>
                  <data>4050,  37.11,  93.74,      </data>
                  <data>4060,   9.85,     95,  1.33</data>
                  <data>4070,  32.44,  89.19,  1.31</data>
                  <data>4080,  29.03, -99999,  1.32</data>
                  <data>4090,  13.09, -99999,  1.34</data>
            </logData>
      </log>
</logs>
```

Metadata about the entire log (table)

Metadata about individual curves (column heading)

Points (rows)

Curves(columns)

From the startIndex and endIndex elements of the header, we can see that the entire log covers the range of 4050 thru 4090. From the minIndex/maxIndex elements of the four logCurveInfo elements, we see that the Mdepth and ROP curves cover this same range. The Bit RPM curve, however, contains non-null data only for 4050-4070, and the ECD curve contains non-null data only for 4060-4090.

The values for the four curves are stored in the log as four data-nodes within each <data> element—one data-node for each curve. (**NOTE:** Additional spaces have been inserted into the example <data> elements to make the example more easily readable. Spaces would not be present between numeric values in actual <data> elements.)

```
…
      <logData>
            <mnemonicList>Mdepth,ROP,Bit RPM,ECD</mnemonicList>
            <unitList>ft,ft/hr,rpm,g/cm3</unitList>
            <data>4050,  37.11,  93.74,      </data>
            <data>4060,   9.85      95,  1.33</data>
            <data>4070,  32.44,  89.19,  1.31</data>
            <data>4080,  29.03, -99999,  1.32</data>
            <data>4090,  13.09, -99999,  1.34</data>
      </logData>
</log>
```

Points

Curves

The data for our four curves happen to be in the same order (defined by mnemonicList) as shown in the header and so the first “column” in the <data> element contains the Mdepth values (4050, 4060, etc) followed by the ROP values in column 2 (37.11, 9.85, etc), the Bit RPM values in column 3 and the ECD values in column 4. A mnemonic cannot exist in the mnemonicList if it does not also exist in the header. The unit of measure in the unitList should match the unit in the curve definition.

If a log curve has no value at a particular depth or time, the value for that column in that <data> element will be the value of the nullValue element specified for that curve or—if no nullValue is specified, a null string.

For example, the Bit RPM curve specifies a value of -99999 for its nullValue element. Therefore, the Bit RPM (3rd) column of the <data> element uses a value of -99999 for the fourth and fifth points to indicate null values. However, since the nullValue element was not specified for the other log curves, a null string (consecutive commas) will be used to indicate null values for all other curves. This is the case with the ECD (4th) column, in which a null string is used to indicate a null value for the first point.

Viewed another way, the five example <data> elements convey the following data points for our five curves:

	Point				
	#1	**#2**	**#3**	**#4**	**#5**
Mdepth	4050	4060	4070	4080	4090
ROP	37.11	9.85	32.44	29.03	13.09
Bit RPM	93.74	95	89.19	-99999	-99999
ECD	null-string	1.33	1.31	1.32	1.34

One of the curves in a log will be designated as the “Index Curve.”

- If the log is depth-based, then the Index Curve is the curve containing the depth at which the values were taken.
- If it is a time-based log, then the Index Curve is the curve containing the time of the measurement.
- If the index mnemonic is listed in the mnemonicList, then it will be the first mnemonic in the list.

Our example is a depth-based log, and Mdepth (curve/column #1) is the Index Curve, in feet. Therefore, the data points represent values taken every 10 feet from 4050 through 4090 feet.

14.2.4.3 Querying for Rows in a Systematically Growing data-object

Now let’s look at the special handling that allows a range of one or more rows to be queried in a systematically growing data-object.

- First, the column(s) to be returned are specified by including zero or more column-identifiers in the data-column-list. For the log data-object, this is the mnemonicList element. If the mnemonicList is empty then all curves will be requested with the index curve being returned first.

- Second, at least one data-node will be included. For the log data-object, this is the <data> element. Specifying an empty data-node will return all data-nodes within the structural-range.
- Third, the rows to be returned are specified by including a structural-range. For the log data-object, this is the elements startIndex and endIndex. If no structural-range is specified then all rows are requested to be returned.

The structural-range applies to all the columns specified in the query.

If there are no rows with node-index values within the specified range of values for start index and end index, then nothing is returned because not all criteria have been met.

Special Handling Summary: log data-object (logData element present)

When the STORE Interface receives a query template for a log data-object, if it contains values for the **startIndex** and **endIndex** elements, and the <logData> and <data> elements are present, then STORE interprets this to mean return the data values for all the specified log curves in the specified range.

Example

The following example shows how to query the ROP and the Bit RPM curves in our example log data-object for data in the range of 4050 thru 4080. (**NOTE:** Because the Bit RPM curve in the example contains data only up to interval 4070, this query results in null values being returned for the Bit RPM curve in the intervals above 4070.)

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
    <log uidWell="W-12" uidWellbore="B-01" uid="L001">
            <startIndex uom="ft">4050</startIndex>
            <endIndex uom="ft">4080</endIndex>
            <logData>
                  <mnemonicList>ROP,Bit RPM</mnemonicList>
                  <unitList></unitList>
                  <data/>
      </logData>
    </log>
</logs>
```

Query template requesting the ROP and Bit RPM curves from 4050 through 4080

If a query specifies values for the min/maxIndex elements in the logCurveInfo element, then those values are ignored:

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
    <log uidWell="W-12" uidWellbore="B-01" uid="L001">
      <startIndex uom="ft">4050</startIndex>
      <endIndex uom="ft">4080</endIndex>
      <logCurveInfo>
            <mnemonic>ROP</mnemonic>
            <minIndex uom="ft">4060</minIndex>      [ignored]
            <maxIndex uom="ft">4070</maxIndex>      [ignored]
      </logCurveInfo>
    </log>
</logs>
```

Query template specifying extraneous min/maxIndex values under logCurveInfo

The previous query is interpreted as if it had specified the example code below. Therefore, the server returns its persisted min/maxIndex values for the ROP curve within the 4050-4080 range.

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
    <log uidWell="W-12" uidWellbore="B-01" uid="L001">
       <startIndex uom="ft">4050</startIndex>
       <endIndex uom="ft">4080</endIndex>
       <logCurveInfo>
             <mnemonic>ROP</mnemonic>
             <minIndex uom=""></minIndex>
             <maxIndex uom=""></maxIndex>
       </logCurveInfo>
    </log>
</logs>
```

Here is another example query requesting the data for our example ROP and Bit RPM curves for the interval range 4060-4100 (inclusive):

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
    <log>
       <startIndex uom="ft">4060</startIndex>
       <endIndex uom="ft">4100</endIndex>
       <logData>
             <mnemonicList>Mdepth,Bit RPM,ECD</mnemonicList>
             <unitList/>
             <data/>
       </logData>
    </log>
</logs>
```

Here is what this query would return. Recalling that our stored log data-object contains:

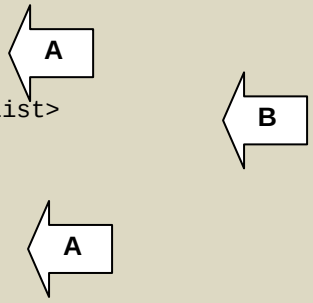
Mdepth	Bit RPM	ECD
<data>4050	93.74	</data>
<data>4060	95.00	1.33</data>
<data>4070	89.19	1.31</data>
<data>4080	-99999	1.32</data>
<data>4090	-99999	1.34</data>

And we specified the range 4060 through 4100 for these three curves:

- Mdepth RPM (curve/column #1 in the stored data)
- Bit RPM (curve/column #3 in the stored data)
- EC (curve/column #4 in the stored data)

The returned document contains:

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
    <log>
      <startIndex uom="ft">4060</startIndex>
      <endIndex uom="ft">4090</endIndex>
      <logData>
            <mnemonicList>Mdepth,Bit RPM,ECD</mnemonicList>
            <unitList>ft,rpm,g/cm3</unitList>
            <data>4060,     95,   1.33</data>
            <data>4070,  89.19,   1.31</data>
            <data>4080, -99999,   1.32</data>
            <data>4090, -99999,   1.34</data>
      </logData>
    </log>
</logs>
```



Results Explanation

This section explains why the query returned the results it did. Letters below refer to the previous figure.

A - The returned start/endIndex values encompass the range of values in the returned curves (not necessarily the entire range of values in the persisted log, nor the range of values that was requested in the query).

In our example, we requested the range of 4060-4100. However, none of the rows for the requested non-index curves contained an index greater than 4090. To maintain the internal consistency of the returned log data-object, the returned start/endIndex values are adjusted to reflect the range actually returned in the curves:

queried for:	<startIndex uom="ft">4060</startIndex> <endIndex uom="ft">4100</endIndex>
returned:	<startIndex uom="ft">4060</startIndex> <endIndex uom="ft">**4090**</endIndex> [adjusted]

B - While the Bit RPM curve data was originally listed as curve #3, and ECD data as curve #4, these relative index values are only relative to the other curves being returned and are not persisted. For example, when the Bit RPM and ECD curves' data values are returned to the client, the curves may have been assigned a new order relative to the other curves returned by that particular query.

In our example, while the Bit RPM curve data was originally listed as curve #3, it is returned as curve #2. Likewise, the ECD data was listed as curve #4, but is returned as curve #3. The server adjusts the order of the returned curves to reflect the order within the returned <data> elements. The client uses the returned column order from that particular query to properly interpret the returned <data> elements. A server may return the curves in a different order on each query.

We can request all curves by using an empty mnemonicList. The following query requests the entire data content of the log. This query presumes that we had previously retrieved the full contents of the curve definitions so that we could properly interpret the curve semantics.

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
     <log uidWell="W-12" uidWellbore="B-01" uid="L001">
       <logData>
             <mnemonicList></mnemonicList>
             <unitList></unitList>
             <data></data>
       </logData>
     </log>
</logs>
```

14.2.4.4 Deleting Rows in a Systematically Growing Data-object

Deleting rows for a systematically growing data-object requires special handling. Specifying a start index or an end index in the header results in the corresponding rows being deleted.

For example, this template deletes all rows where the indexCurve value is greater than or equal to 4060 and the indexCurve value is less than or equal to 4080:

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
      <log uidWell="W-12" uidWellbore="B-01" uid="f34a">
            <startIndex uom="ft">4060</startIndex>
            <endIndex uom="ft">4080</endIndex>
      </log>
</logs>
```

…and results in this data being left on the server:

<data>4050	37.11	93.74	</data>
<data>4090	13.09	-99999	1.34</data>

The above example can be modified to only delete data for specific mnemonics, for example:

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
      <log uidWell="W-12" uidWellbore="B-01" uid="f34a">
            <startIndex uom="ft">4060</startIndex>
            <endIndex uom="ft">4080</endIndex>
            <logCurveInfo uid="rop">
                  <mnemonic>ROP</mnemonic>
            </logCurveInfo>
            <logCurveInfo uid="bit_rpm">
                  <mnemonic>Bit RPM</mnemonic>
            </logCurveInfo>
      </log>
</logs>
```

… which results in this data being left on the server.

<data>4050	37.11	93.74	</data>
<data>4060			1.33</data>
<data>4070			1.31</data>
<data>4080			1.32</data>
<data>4090	13.09	-99999	1.34</data>

The index curve cannot be deleted unless the whole row is deleted. As shown in the first example, if specific mnemonics had not been specified, then the whole row would have been deleted.

- Specifying only startIndex deletes from (and including) that point to the maximum index of the log.
- Specifying only endIndex deletes from the minimum index of the log to (and including) endIndex. The startIndex and endIndex will be updated after the delete for all affected curves.

14.2.4.5 Adding Columns in a Systematically Growing data-object

The general STORE behavior for updating a data-object with data that does not yet exist is to add the data using WMLS_UpdateInStore. No special behavior is required to update a systematically growing data-object with a new column, but the row index must also be specified. The header definition of the column must also be defined when or before the data is added. New columns may be added or existing columns updated in the same update call.

This template:

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
    <log uidWell="W-12" uidWellbore="B-01" uid="L001">
      <logCurveInfo uid="HLKD">
            <mnemonic>HKLD</mnemonic>
            <unit>klbf</unit>
      </logCurveInfo>
      <logData>
            <mnemonicList>Mdepth,HKLD</mnemonicList>
            <unitList>ft,klbf</unitList>
            <data>4050,187.66</data>
            <data>4060,185.74</data>
            <data>4070,184.23</data>
            <data>4080,185.49</data>
            <data>4090,185.55</data>
      </logData >
    </log>
</logs>
```

Update adding one column to a particular log.

… results in this data in the server:

<data>4050	37.11	93.74		**187.66**</data>
<data>4060	9.85	95	1.33	**185.74**</data>
<data>4070	32.44	89.19	1.31	**184.23**</data>
<data>4080	29.03	-99999	1.32	**185.49**</data>
<data>4090	13.09	-99999	1.34	**185.55**</data>

14.2.4.6 Deleting Columns in a Systematically Growing Data-object

To delete a column, specify its column-identifier in the header of the delete template (the QueryIn section of the WMLS_DeleteFromStore function).

Thus, this template:

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
    <log uidWell="W-12" uidWellbore="B-01" uid="L001">
      <logCurveInfo>
            <mnemonic>ECD</mnemonic>
      </logCurveInfo>
    </log>
</logs>
```

…results in this data in the server:

<data>4050	37.11	93.74</data>
<data>4060	9.85	95</data>
<data>4070	32.44	89.19</data>
<data>4080	29.03	-99999</data>
<data>4090	13.09	-99999</data>

14.2.5 Decreasing Logs

Decreasing logs are certainly not the 80% case, but they are gaining in importance as wireline and multi-pass logging become more prevalent in WITMSL. Of course, decreasing logs are only relevant for depth indexes, not time.

A decreasing log goes through the same birth-to-retirement cycle as an increasing log:

1. It is **created** on the server via WMLS_AddToStore
2. It is **read** via WMLS_GetFromStore
3. It is **updated** and/or appended via WMLS_UpdateInStore
4. It is **retired** via WMLS_DeleteFromStore

Although 1 and 4 are single events, 2 and 3 can occur repetitively, and in any order.

The following sections show an example of how a decreasing log is expected to behave, in particular for 1 and 2.

14.2.5.1 Creating a Decreasing Log

A decreasing log is created using the WMLS_AddToStore function. The important things are that:

- The <direction> attribute is set to **decreasing** in the request and on the server.
- The <startIndex> and <endIndex> elements are reversed.
- The <data> rows are ordered from deeper to shallower.

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
  <log  uidWell="OC-bf" uidWellbore="OC-bf-wb1" uid="DL-1">
    <nameWell>OC Test Bruce</nameWell>
    <nameWellbore>Wellbore Test Bruce</nameWellbore>
    <name>DecreasingLog</name>
    <runNumber>1</runNumber>
    <pass>ReamUp2</pass>
    <creationDate>2011-06-23T15:51:51.456</creationDate>
    <indexType>measured depth</indexType>
    <startIndex uom="ft">1000</startIndex>
    <endIndex uom="ft">998</endIndex>
    <direction>decreasing</direction>
    <indexCurve columnIndex="1">Depth</indexCurve>
    <nullValue>-999.25</nullValue>
    <logCurveInfo>
      <mnemonic>Depth</mnemonic>
      <unit>ft</unit>
      <mnemAlias>md</mnemAlias>
      <nullValue>-999.25</nullValue>
      <columnIndex>1</columnIndex>
      <curveDescription>measured depth index</curveDescription>
      <typeLogData>double</typeLogData>
    </logCurveInfo>
    <logCurveInfo>
      <mnemonic>Channel1</mnemonic>
      <unit>psi</unit>
      <mnemAlias>Channel1</mnemAlias>
      <nullValue>-999.25</nullValue>
      <columnIndex>2</columnIndex>
      <curveDescription>Channel1</curveDescription>
      <typeLogData>double</typeLogData>
    </logCurveInfo>
    <logCurveInfo>
      <mnemonic>Channel2</mnemonic>
      <unit>deg</unit>
      <mnemAlias>Channel2</mnemAlias>
      <nullValue>-999.25</nullValue>
      <columnIndex>3</columnIndex>
      <curveDescription>Channel2</curveDescription>
      <typeLogData>double</typeLogData>
    </logCurveInfo>
    <logData>
      <data>1000,1,2</data>
      <data>999,6,7</data>
      <data>998,12,11</data>
    </logData>
  </log>
</logs>
```

14.2.5.2 Reading a Decreasing Log

Reading Log Meta Data

In the following request and response for log meta-data, the expected pattern is that, in the response:

- The <direction> is returned as *decreasing*,
- <startIndex> and <endIndex> are *reversed*, and
- <minIndex> and <maxIndex> are the *mathematical min/max index* of each log curve.

Request

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
  <log uidWell="OC-bf" uidWellbore="OC-bf-wb1" uid="DL-1">
    <direction />
    <startIndex />
    <endIndex />
    <logCurveInfo uid="">
      <mnemonic />
      <minIndex uom="" />
      <maxIndex uom="" />
    </logCurveInfo>
  </log>
</logs>
```

Response

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
  <log uidWell="OC-bf" uidWellbore="OC-bf-wb1" uid="DL-1">
    <startIndex>1000</startIndex>
    <endIndex>998</endIndex>
    <direction>decreasing</direction>
    <logCurveInfo uid="Channel2">
      <mnemonic>Channel2</mnemonic>
      <minIndex uom="ft">998</minIndex>
      <maxIndex uom="ft">1000</maxIndex>
    </logCurveInfo>
    <logCurveInfo uid="Depth">
      <mnemonic>Depth</mnemonic>
      <minIndex uom="ft">998</minIndex>
      <maxIndex uom="ft">1000</maxIndex>
    </logCurveInfo>
    <logCurveInfo uid="Channel1">
      <mnemonic>Channel1</mnemonic>
      <minIndex uom="ft">998</minIndex>
      <maxIndex uom="ft">1000</maxIndex>
    </logCurveInfo>
  </log>
</logs>
```

14.2.5.3 Reading Log Data

The response is expected to have the <startIndex>/<endIndex> **reversed**, and sequence the <data> rows in *reversed* order.

Request

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
  <log uidWell="OC-bf" uidWellbore="OC-bf-wb1" uid="DL-1">
    <indexType>measured depth</indexType>
    <startIndex>1000</startIndex>
    <endIndex></endIndex>
    <indexCurve columnIndex="" />
    <logCurveInfo>
      <mnemonic>Depth</mnemonic>
      <columnIndex />
      <unit />
      <typeLogData />
    </logCurveInfo>
    <logCurveInfo>
      <mnemonic>Channel2</mnemonic>
      <columnIndex />
      <unit />
      <typeLogData />
    </logCurveInfo>
    <logCurveInfo>
      <mnemonic>Channel1</mnemonic>
      <columnIndex />
      <unit />
      <typeLogData />
    </logCurveInfo>
    <logData>
      <data />
    </logData>
  </log>
</logs>
```

Response

```
<logs xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
  <log uidWell="OC-bf" uidWellbore="OC-bf-wb1" uid="DL-1">
    <indexType>measured depth</indexType>
    <startIndex>1000</startIndex>
    <endIndex>998</endIndex>
    <indexCurve columnIndex="1">Depth</indexCurve>
    <logCurveInfo>
      <mnemonic>Channel2</mnemonic>
      <unit>deg</unit>
      <columnIndex>2</columnIndex>
      <typeLogData>double</typeLogData>
    </logCurveInfo>
    <logCurveInfo>
      <mnemonic>Channel1</mnemonic>
      <unit>psi</unit>
      <columnIndex>3</columnIndex>
      <typeLogData>double</typeLogData>
    </logCurveInfo>
    <logCurveInfo>
      <mnemonic>Depth</mnemonic>
      <unit>ft</unit>
      <columnIndex>1</columnIndex>
      <typeLogData>double</typeLogData>
    </logCurveInfo>
    <logData>
      <data>1000,2,1</data>
      <data>999,7,6</data>
```

```
      <data>998,12,11</data>
    </logData>
  </log>
</logs>
```

14.2.6 groupedObject Query Examples

The first example (Query 1) is to query a server for the data for a group of elapsed time logs from well testing in a specified well and wellbore.

Query 1

```
<?xml version="1.0" encoding="utf-8"?>
<objectGroups xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
   <objectGroup uidWell="x21345abc" uidWellbore="x21345abc" uid="">
      <groupType>pressure test</groupType>
      <memberObject uid="">
         <objectReference object="log" uidRef=""></objectReference>
         <indexType>elapsed time</indexType>
         <sequence1 description=""/>
         <sequence2 description=""/>
         <sequence3 description=""/>
         <rangeMin uom="" />
         <rangeMax uom="" />
         <mnemonicList />
         <referenceDepth uom="" />
        <referenceDateTime />
      </memberObject>
      <commonData>
         <dTimCreation>2010-12-12T00:00:00Z</dTimCreation>
      </commonData>
   </objectGroup>
</objectGroups>
```

XMLOut

```
<?xml version="1.0" encoding="utf-8"?>
<objectGroups xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
   <objectGroup uidWell="x21345abc" uidWellbore="x21345abc" uid="x21345logs">
      <groupType>pressure test</groupType>
      <nameWell></nameWell>
      <nameWellbore></nameWellbore>
      <name></name>
      <memberObject uid="presstest1">
         <objectReference object="log" uidRef="ptestlog1">Pressure Test
1</objectReference>
         <indexType>elapsed time</indexType>
         <sequence1 description="order">1</sequence1>
         <rangeMin uom="s">0</rangeMin>
         <rangeMax uom="s">900</rangeMax>
         <mnemonicList>ELTIME,PRES,TEMP</mnemonicList>
         <referenceDepth uom="ft">7807.5</referenceDepth>
         <referenceDateTime>2011-08-26T12:45:16Z</referenceDateTime>
      </memberObject>
      <memberObject uid="presstest2">
         <objectReference object="log" uidRef="ptestlog2">Pressure Test
2</objectReference>
         <indexType>elapsed time</indexType>
        <sequence1 description="order">2</sequence1>
         <rangeMin uom="s">0</rangeMin>
         <rangeMax uom="s">1150</rangeMax>
         <mnemonicList>ELTIME,PREiS,TEMP,DENSITY</mnemonicList>
         <referenceDepth uom="ft">7854.45</referenceDepth>
         <referenceDateTime>2011-08-26T22:32:48Z</referenceDateTime>
      </memberObject>
      <commonData>
         <dTimCreation>2011-08-27T00:00:00Z</dTimCreation>
```

```
      </commonData>
   </objectGroup>
</objectGroups>
```

Query 2 shows how to query when the object groupType equals “other”. In this case, the subGroupType (free-form text) must also be used to specify the name of the “other” groupType.

Query 2

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  This example is non-normative and is only intended to demonstrate the type of data
that can be exchanged.
  It is totally artificial and is not intended to demonstrate best practices.
-->
<objectGroups
 xmlns="http://www.witsml.org/schemas/1series"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.witsml.org/schemas/1series
../xsd_schemas/obj_objectGroup.xsd"
 version="1.4.1.1">
 <objectGroup uidWell="W-12" uidWellbore="B-01" uid="8C955975-BCE2-44D8-9C8F-
5B78F7F13F21">
  <nameWell>6507/7-A-42</nameWell>
  <nameWellbore>A-42</nameWellbore>
  <name>GeoSteering Objects</name>
  <groupType>other</groupType>
  <groupSubType>GeoSteering</groupSubType>
  <description>Objects pertaining to GeoSteering</description>
  <param index="1" name="MRES" uom="ohm.m" description="Mud
Resistivity">1.25</param>
  <param index="2" name="BDIA" uom="in" description="Bit Diameter">12.25</param>
  <memberObject uid="M-001">
   <objectReference object="log" uidRef="f34a">L001</objectReference>
   <indexType>measured depth</indexType>
   <sequence1 description="Log">1</sequence1>
   <rangeMin uom="m">499</rangeMin>
   <rangeMax uom="m">4201.23</rangeMax>
   <mnemonicList>Mdepth,Vdepth,ROP,GR,RES1,RES2,AZIDEN</mnemonicList>
  </memberObject>
  <memberObject uid="M-002">
   <objectReference object="mudLog" uidRef="h45a">m1001</objectReference>
   <sequence1 description="mudLog">2</sequence1>
   <rangeMin uom="m">499</rangeMin>
   <rangeMax uom="m">4201.23</rangeMax>
  </memberObject>
  <memberObject uid="M-003">
   <objectReference object="trajectory" uidRef="h45a">pe84e</objectReference>
   <sequence1 description="trajectory">3</sequence1>
   <rangeMin uom="m">499</rangeMin>
   <rangeMax uom="m">4201.23</rangeMax>
  </memberObject>
  <commonData>
   <sourceName>Geologix</sourceName>
   <dTimCreation>2012-04-19T10:23:45Z</dTimCreation>
   <dTimLastChange>2012-04-29T10:23:45Z</dTimLastChange>
   <itemState>actual</itemState>
  </commonData>
  <customData />
 </objectGroup>
</objectGroups>
```

14.3 Example Output from Standard Query 9

This output is from an example for Standard Query 9 (SQ-009) found in Section 6.6.5.10, page 76. The example output is very long so it was moved to this appendix.

```
<opsReports xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1">
    <opsReport uidWell="UUID-1" uidWellbore="wellbore" uid="ops-0001">
        <nameWell>UUIDWell</nameWell>
        <nameWellbore>Wellbore 1234</nameWellbore>
        <name>ops-0001</name>
        <rig uidRef="H56p">Big Rig #5</rig>
        <dTim>2001-11-01T08:15:00.000Z</dTim>
        <eTimStart uom="d">19</eTimStart>
        <eTimSpud uom="d">18</eTimSpud>
        <eTimLoc uom="d">20</eTimLoc>
        <mdReport uom="m">2339</mdReport>
        <tvdReport uom="m">2113.4</tvdReport>
        <distDrill uom="m">46</distDrill>
        <eTimDrill uom="h">120</eTimDrill>
        <mdPlanned uom="m">2640</mdPlanned>
        <ropAv uom="m/h">17.8</ropAv>
        <ropCurrent uom="m/h">14.6</ropCurrent>
        <supervisor>Mike Taylor</supervisor>
        <engineer>Anton Gerbrady</engineer>
        <geologist>Andrew Williams</geologist>
        <eTimDrillRot uom="h">12</eTimDrillRot>
        <eTimDrillSlid uom="h">0</eTimDrillSlid>
        <eTimCirc uom="h">3</eTimCirc>
        <eTimReam uom="h">0</eTimReam>
        <eTimHold uom="h">0</eTimHold>
        <eTimSteering uom="h">0</eTimSteering>
        <distDrillRot uom="m">46</distDrillRot>
        <distDrillSlid uom="m">0</distDrillSlid>
        <distReam uom="m">0</distReam>
        <distHold uom="m">0</distHold>
        <distSteering uom="m">0</distSteering>
        <numPob>99</numPob>
        <numContract>48</numContract>
        <numOperator>1</numOperator>
        <numService>50</numService>
        <activity uid="act-1">
            <dTimStart>2001-10-24T06:00:00.000Z</dTimStart>
            <dTimEnd>2001-10-31T09:15:00.000Z</dTimEnd>
            <duration uom="h">3.25</duration>
            <phase>Int2</phase>
            <activityCode>tripping</activityCode>
            <detailActivity>No detail</detailActivity>
            <typeActivityClass>planned</typeActivityClass>
            <mdHoleStart uom="ft">7000</mdHoleStart>
            <tvdHoleStart uom="m">6999</tvdHoleStart>
            <mdHoleEnd uom="ft">7000</mdHoleEnd>
            <tvdHoleEnd uom="ft">6999</tvdHoleEnd>
            <mdBitStart uom="ft">6999</mdBitStart>
            <mdBitEnd uom="ft">0</mdBitEnd>
            <state>F</state>
            <operator>Petoc Resources</operator>
            <optimum>false</optimum>
            <productive>true</productive>
            <itemState>actual</itemState>
            <comments>Continued pulling and racking BHA in derrick. Break off
bit.</comments>
        </activity>
        <drillingParams uid="dp-1">
            <eTimOpBit uom="h">2.25</eTimOpBit>
            <mdHoleStart uom="ft">6000</mdHoleStart>
            <mdHoleStop uom="ft">7000</mdHoleStop>
            <tubular uidRef="TUB-23">12.25-R6</tubular>
```

```
            <hkldRot uom="tonfUS">150000</hkldRot>
            <overPull uom="tonfUS">230000</overPull>
            <slackOff uom="tonfUS">180000</slackOff>
            <hkldUp uom="tonfUS">300000</hkldUp>
            <hkldDn uom="tonfUS">300000</hkldDn>
            <tqOnBotAv uom="ft.lbf">3000</tqOnBotAv>
            <tqOnBotMx uom="ft.lbf">3500</tqOnBotMx>
            <tqOnBotMn uom="ft.lbf">2800</tqOnBotMn>
            <tqOffBotAv uom="ft.lbf">3000</tqOffBotAv>
            <tqDhAv uom="ft.lbf">3000</tqDhAv>
            <wtAboveJar uom="lbf">300000</wtAboveJar>
            <wtBelowJar uom="lbf">300000</wtBelowJar>
            <wtMud uom="lbm/ft3">1.24</wtMud>
            <flowratePump uom="ft3/h">400</flowratePump>
            <powBit uom="hp">500</powBit>
            <velNozzleAv uom="ft/s">15</velNozzleAv>
            <presDropBit uom="psi">500</presDropBit>
            <cTimHold uom="h">0.3</cTimHold>
            <cTimSteering uom="h">1.5</cTimSteering>
            <cTimDrillRot uom="h">2.25</cTimDrillRot>
            <cTimDrillSlid uom="h">0.45</cTimDrillSlid>
            <cTimCirc uom="h">2.3</cTimCirc>
            <cTimReam uom="h">1.25</cTimReam>
            <distDrillRot uom="ft">300</distDrillRot>
            <distDrillSlid uom="ft">200</distDrillSlid>
            <distReam uom="ft">50</distReam>
            <distHold uom="ft">100</distHold>
            <distSteering uom="ft">200</distSteering>
            <rpmAv uom="rpm">100</rpmAv>
            <rpmMx uom="rpm">110</rpmMx>
            <rpmMn uom="rpm">80</rpmMn>
            <rpmAvDh uom="rpm">80</rpmAvDh>
            <ropAv uom="ft/h">80</ropAv>
            <ropMx uom="ft/h">90</ropMx>
            <ropMn uom="ft/h">70</ropMn>
            <wobAv uom="tonfUS">12.5</wobAv>
            <wobMx uom="tonfUS">15</wobMx>
            <wobMn uom="tonfUS">11</wobMn>
            <wobAvDh uom="tonfUS">8.5</wobAvDh>
            <reasonTrip>Coring point</reasonTrip>
            <objectiveBha>Drilling</objectiveBha>
            <aziTop uom="dega">30</aziTop>
            <aziBottom uom="dega">45</aziBottom>
            <inclStart uom="dega">0</inclStart>
            <inclMx uom="dega">20</inclMx>
            <inclMn uom="dega">0</inclMn>
            <inclStop uom="dega">10</inclStop>
            <tempMudDhMx uom="degC">92</tempMudDhMx>
            <presPumpAv uom="psi">3000</presPumpAv>
            <flowrateBit uom="bbl/min">60</flowrateBit>
            <comments>Hi TQ spikes through stringers</comments>
        </drillingParams>
        <dayCost uid="dc-2">
            <numAFE>1234-5678</numAFE>
            <costGroup>NA</costGroup>
            <costClass>TAN</costClass>
            <costCode>2</costCode>
            <costSubCode>14</costSubCode>
            <costItemDescription>Rig fuel cost</costItemDescription>
            <costPerItem currency="USD">1.42</costPerItem>
            <itemKind>gal</itemKind>
            <itemSize>3000</itemSize>
            <qtyItem>1</qtyItem>
            <costAmount currency="USD">1420</costAmount>
            <numInvoice>261002</numInvoice>
            <numPO>570013</numPO>
```

```
            <numTicket>None</numTicket>
            <isCarryOver>false</isCarryOver>
            <isRental>false</isRental>
            <numSerial>N/A</numSerial>
            <nameVendor>Offshore Gas Supply</nameVendor>
            <numVendor>98730</numVendor>
            <pool>NA</pool>
            <estimated>true</estimated>
        </dayCost>
        <dayCost uid="dc-1">
            <numAFE>1234-5678</numAFE>
            <costGroup>NA</costGroup>
            <costClass>TAN</costClass>
            <costCode>2</costCode>
            <costSubCode>13</costSubCode>
            <costItemDescription>Rig dayrate</costItemDescription>
            <costPerItem currency="USD">30000</costPerItem>
            <itemKind>day</itemKind>
            <itemSize>1</itemSize>
            <qtyItem>1</qtyItem>
            <costAmount currency="USD">30000</costAmount>
            <numInvoice>23-261001</numInvoice>
            <numPO>1-570012</numPO>
            <numTicket>12345</numTicket>
            <isCarryOver>true</isCarryOver>
            <isRental>false</isRental>
            <numSerial>12345</numSerial>
            <nameVendor>Transocean Sedco-Forex</nameVendor>
            <numVendor>98765</numVendor>
            <pool>NA</pool>
            <estimated>true</estimated>
        </dayCost>
        <trajectoryStation uid="34ht5">
            <dTimStn>2001-10-21T08:15:00.000Z</dTimStn>
            <typeTrajStation>tie in point</typeTrajStation>
            <md uom="ft">6000</md>
            <tvd uom="ft">5556</tvd>
            <incl uom="dega">10</incl>
            <azi uom="dega">47.3</azi>
            <mtf uom="dega">47.3</mtf>
            <gtf uom="dega">0</gtf>
            <dispNs uom="ft">200</dispNs>
            <dispEw uom="ft">345</dispEw>
            <vertSect uom="ft">378</vertSect>
            <dls uom="dega/ft">1.5</dls>
            <rateTurn uom="dega/ft">0.8</rateTurn>
            <rateBuild uom="dega/ft">1.9</rateBuild>
            <mdDelta uom="ft">30</mdDelta>
            <tvdDelta uom="ft">29.9</tvdDelta>
            <modelToolError>good gyro</modelToolError>
            <gravTotalUncert uom="m/s2">0</gravTotalUncert>
            <dipAngleUncert uom="dega">0.1</dipAngleUncert>
            <magTotalUncert uom="mT">0</magTotalUncert>
            <gravAccelCorUsed>false</gravAccelCorUsed>
            <magXAxialCorUsed>false</magXAxialCorUsed>
            <sagCorUsed>false</sagCorUsed>
            <magDrlstrCorUsed>false</magDrlstrCorUsed>
            <statusTrajStation>position</statusTrajStation>
            <rawData>
                <gravAxialRaw uom="ft/s2">0.116</gravAxialRaw>
                <gravTran1Raw uom="ft/s2">-0.168</gravTran1Raw>
                <gravTran2Raw uom="ft/s2">-1654</gravTran2Raw>
                <magAxialRaw uom="mT">22.77</magAxialRaw>
                <magTran1Raw uom="mT">22.5</magTran1Raw>
                <magTran2Raw uom="mT">27.05</magTran2Raw>
            </rawData>
```

```
            <corUsed>
                <gravAxialAccelCor uom="ft/s2">0.11</gravAxialAccelCor>
                <gravTran1AccelCor uom="ft/s2">0.14</gravTran1AccelCor>
                <gravTran2AccelCor uom="ft/s2">0.13</gravTran2AccelCor>
                <magAxialDrlstrCor uom="mT">0.17</magAxialDrlstrCor>
                <magTran1DrlstrCor uom="mT">0.16</magTran1DrlstrCor>
                <magTran2DrlstrCor uom="mT">0.24</magTran2DrlstrCor>
                <sagIncCor uom="dega">0</sagIncCor>
                <sagAziCor uom="dega">0</sagAziCor>
                <stnMagDeclUsed uom="dega">-4.038</stnMagDeclUsed>
                <stnGridCorUsed uom="dega">-0.4917</stnGridCorUsed>
                <dirSensorOffset uom="ft">48.3</dirSensorOffset>
            </corUsed>
            <valid>
                <magTotalFieldCalc uom="mT">51.19</magTotalFieldCalc>
                <magDipAngleCalc uom="dega">41.5</magDipAngleCalc>
                <gravTotalFieldCalc uom="ft/s2">0.999</gravTotalFieldCalc>
            </valid>
            <matrixCov>
                <varianceNN uom="ft2">100</varianceNN>
                <varianceNE uom="ft2">144</varianceNE>
                <varianceNVert uom="ft2">121</varianceNVert>
                <varianceEE uom="ft2">144</varianceEE>
                <varianceEVert uom="ft2">121</varianceEVert>
                <varianceVertVert uom="ft2">100</varianceVertVert>
                <biasN uom="ft">8</biasN>
                <biasE uom="ft">4</biasE>
                <biasVert uom="ft">2</biasVert>
            </matrixCov>
            <location uid="location-2">
                <wellCRS uidRef="crs1">AlbersEqualArea</wellCRS>
                <projectedX uom="ft">500</projectedX>
                <projectedY uom="ft">300</projectedY>
            </location>
            <location uid="location-1">
                <wellCRS uidRef="crs2">Adindian</wellCRS>
                <latitude uom="dega">93</latitude>
                <longitude uom="dega">42</longitude>
            </location>
        </trajectoryStation>
        <fluid uid="fluid-1">
            <type>oil based</type>
            <locationSample>Pit</locationSample>
            <density uom="g/L">1.26</density>
            <visFunnel uom="s">55</visFunnel>
            <tempVis uom="degC">28</tempVis>
            <pv uom="cP">50</pv>
            <yp uom="lbf/100ft2">30</yp>
            <gel10Sec uom="lbf/100ft2">14</gel10Sec>
            <gel10Min uom="lbf/100ft2">24</gel10Min>
            <gel30Min uom="lbf/100ft2">27</gel30Min>
            <filterCakeLtlp uom="in/32">1</filterCakeLtlp>
            <filtrateLtlp uom="mL">2.3</filtrateLtlp>
            <tempHthp uom="degF">200</tempHthp>
            <presHthp uom="psi">500</presHthp>
            <filtrateHthp uom="mL">2.4</filtrateHthp>
            <filterCakeHthp uom="in/32">1</filterCakeHthp>
            <solidsPc uom="%">6.1</solidsPc>
            <waterPc uom="%">32</waterPc>
            <oilPc uom="%">53</oilPc>
            <sandPc uom="%">0.3</sandPc>
            <solidsLowGravPc uom="%">6.1</solidsLowGravPc>
            <solidsCalcPc uom="%">7.5</solidsCalcPc>
            <baritePc uom="%">5</baritePc>
            <lcm uom="lbm/bbl">10</lcm>
            <mbt uom="meq/g">0.3</mbt>
```

```
            <ph>10.5</ph>
            <tempPh uom="degC">28</tempPh>
            <pm uom="mL">0.9</pm>
            <pmFiltrate uom="mL">1.3</pmFiltrate>
            <mf uom="mL">1</mf>
            <alkalinityP1 uom="mL">1.1</alkalinityP1>
            <alkalinityP2 uom="mL">1.2</alkalinityP2>
            <chloride uom="mg/L">18750</chloride>
            <calcium uom="mg/L">7000</calcium>
            <magnesium uom="mg/L">104.5</magnesium>
            <rheometer uid="rheometer-1">
                <tempRheom uom="degC">25</tempRheom>
                <presRheom uom="psi">14.5</presRheom>
                <vis3Rpm>10</vis3Rpm>
                <vis6Rpm>12</vis6Rpm>
                <vis100Rpm>40</vis100Rpm>
                <vis200Rpm>61</vis200Rpm>
                <vis300Rpm>80</vis300Rpm>
                <vis600Rpm>130</vis600Rpm>
            </rheometer>
            <rheometer uid="rheometer-2">
                <tempRheom uom="degC">26</tempRheom>
                <presRheom uom="psi">14.5</presRheom>
                <vis3Rpm>11</vis3Rpm>
                <vis6Rpm>13</vis6Rpm>
                <vis100Rpm>42</vis100Rpm>
                <vis200Rpm>63</vis200Rpm>
                <vis300Rpm>83</vis300Rpm>
                <vis600Rpm>135</vis600Rpm>
            </rheometer>
            <brinePc uom="%">20</brinePc>
            <lime uom="lbm/bbl">3.2</lime>
            <electStab uom="V">360</electStab>
            <calciumChloride uom="mg/L">5000</calciumChloride>
            <company>Baker Hughes INTEQ</company>
            <engineer>Doug Bullivant/Andy Duncombe</engineer>
            <asg>0.0275</asg>
            <solidsHiGravPc uom="%">7.22</solidsHiGravPc>
            <polymer uom="%">2.5</polymer>
            <polyType>XCD POLYMER</polyType>
            <solCorPc uom="%">3.53</solCorPc>
            <oilCtg uom="g/kg">110.3</oilCtg>
            <hardnessCa uom="ppm">3200</hardnessCa>
            <sulfide uom="g/L">10.5</sulfide>
            <comments>Base oil, fluid loss additive and viscosifier to active and
reserve mud.</comments>
        </fluid>
        <scr uid="scr-2">
            <dTim>2001-10-25T14:00:00.000Z</dTim>
            <pump uidRef="PUMP-2">2</pump>
            <typeScr>string annulus</typeScr>
            <rateStroke uom="rpm">30</rateStroke>
            <presRecorded uom="psi">310</presRecorded>
            <mdBit uom="ft">7000</mdBit>
        </scr>
        <scr uid="scr-1">
            <dTim>2001-10-25T06:00:00.000Z</dTim>
            <pump uidRef="PUMP-1">1</pump>
            <typeScr>string annulus</typeScr>
            <rateStroke uom="rpm">20</rateStroke>
            <presRecorded uom="psi">200</presRecorded>
            <mdBit uom="ft">7000</mdBit>
        </scr>
        <pitVolume uid="pv-2">
            <pit uidRef="Active-2">2</pit>
            <dTim>2001-10-25T06:05:00.000Z</dTim>
```

```
            <volPit uom="bbl">160</volPit>
            <densFluid uom="lbm/galUS">10.9</densFluid>
            <descFluid>Mud</descFluid>
            <visFunnel uom="s">36</visFunnel>
        </pitVolume>
        <pitVolume uid="pv-1">
            <pit uidRef="Active-1">1</pit>
            <dTim>2001-10-25T06:00:00.000Z</dTim>
            <volPit uom="bbl">155</volPit>
            <densFluid uom="lbm/galUS">10.8</densFluid>
            <descFluid>Mud</descFluid>
            <visFunnel uom="s">37</visFunnel>
        </pitVolume>
        <mudVolume>
            <volTotMudStart uom="bbl">275</volTotMudStart>
            <volMudDumped uom="bbl">0</volMudDumped>
            <volMudReceived uom="bbl">45</volMudReceived>
            <volMudReturned uom="bbl">70</volMudReturned>
            <mudLosses>
                <volLostShakerSurf uom="bbl">2</volLostShakerSurf>
                <volLostMudCleanerSurf uom="bbl">0.25</volLostMudCleanerSurf>
                <volLostPitsSurf uom="bbl">0</volLostPitsSurf>
                <volLostTrippingSurf uom="bbl">1</volLostTrippingSurf>
                <volLostOtherSurf uom="bbl">0</volLostOtherSurf>
                <volTotMudLostSurf uom="bbl">3.25</volTotMudLostSurf>
                <volLostCircHole uom="bbl">5</volLostCircHole>
                <volLostCsgHole uom="bbl">0</volLostCsgHole>
                <volLostCmtHole uom="bbl">0</volLostCmtHole>
                <volLostBhdCsgHole uom="bbl">0</volLostBhdCsgHole>
                <volLostAbandonHole uom="bbl">0</volLostAbandonHole>
                <volLostOtherHole uom="bbl">0</volLostOtherHole>
                <volTotMudLostHole uom="bbl">5</volTotMudLostHole>
            </mudLosses>
            <volMudBuilt uom="bbl">0</volMudBuilt>
            <volMudString uom="bbl">30</volMudString>
            <volMudCasing uom="bbl">100</volMudCasing>
            <volMudHole uom="bbl">65</volMudHole>
            <volMudRiser uom="bbl">15</volMudRiser>
            <volTotMudEnd uom="bbl">310</volTotMudEnd>
        </mudVolume>
        <mudInventory uid="mi-1">
            <name>ANCO ZAN D</name>
            <itemWtPerUnit uom="kg">25</itemWtPerUnit>
            <pricePerUnit currency="USD">100</pricePerUnit>
            <qtyStart>40</qtyStart>
            <qtyAdjustment>-2</qtyAdjustment>
            <qtyReceived>40</qtyReceived>
            <qtyReturned>0</qtyReturned>
            <qtyUsed>10</qtyUsed>
            <costItem currency="USD">3000</costItem>
            <qtyOnLocation>68</qtyOnLocation>
        </mudInventory>
        <bulk uid="bulk-1">
            <name>Heli Fuel</name>
            <itemVolPerUnit uom="L">1</itemVolPerUnit>
            <pricePerUnit currency="USD">2.5</pricePerUnit>
            <qtyStart>8585</qtyStart>
            <qtyAdjustment>0</qtyAdjustment>
            <qtyReceived>0</qtyReceived>
            <qtyReturned>0</qtyReturned>
            <qtyUsed>9</qtyUsed>
            <costItem currency="USD">250</costItem>
            <qtyOnLocation>8576</qtyOnLocation>
        </bulk>
        <rigResponse>
            <anchorTension index="4" name="Anchor4" uom="lbf">210</anchorTension>
```

```
            <anchorTension index="3" name="Anchor3" uom="lbf">203</anchorTension>
            <anchorTension index="2" name="Anchor2" uom="lbf">205</anchorTension>
            <anchorTension index="1" name="Anchor1" uom="lbf">200</anchorTension>
            <anchorAngle index="3" name="AnchorAngle3" uom="dega">30</anchorAngle>
            <anchorAngle index="4" name="AnchorAngle4" uom="dega">30</anchorAngle>
            <anchorAngle index="1" name="AnchorAngle1" uom="dega">30</anchorAngle>
            <anchorAngle index="2" name="AnchorAngle2" uom="dega">30</anchorAngle>
            <rigHeading uom="dega">100</rigHeading>
            <rigHeave uom="ft">12</rigHeave>
            <rigPitchAngle uom="dega">6</rigPitchAngle>
            <rigRollAngle uom="dega">3</rigRollAngle>
            <riserAngle uom="dega">3</riserAngle>
            <riserDirection uom="dega">10</riserDirection>
            <riserTension uom="klbf">10</riserTension>
            <variableDeckLoad uom="klbf">99</variableDeckLoad>
            <totalDeckLoad uom="klbf">222</totalDeckLoad>
            <guideBaseAngle uom="dega">100</guideBaseAngle>
            <ballJointAngle uom="dega">0</ballJointAngle>
            <ballJointDirection uom="dega">100</ballJointDirection>
            <offsetRig uom="ft">1</offsetRig>
            <loadLeg1 uom="klbf">0</loadLeg1>
            <loadLeg2 uom="klbf">0</loadLeg2>
            <loadLeg3 uom="klbf">0</loadLeg3>
            <loadLeg4 uom="klbf">0</loadLeg4>
            <penetrationLeg1 uom="ft">1.5</penetrationLeg1>
            <penetrationLeg2 uom="ft">1.5</penetrationLeg2>
            <penetrationLeg3 uom="ft">1.5</penetrationLeg3>
            <penetrationLeg4 uom="ft">1.5</penetrationLeg4>
            <dispRig uom="ft">12</dispRig>
            <meanDraft uom="ft">32</meanDraft>
        </rigResponse>
        <pumpOp uid="po-1">
            <dTim>2001-10-25T06:00:00.000Z</dTim>
            <pump uidRef="Pump-1">1</pump>
            <typeOperation>drilling</typeOperation>
            <idLiner uom="in">7</idLiner>
            <lenStroke uom="in">14</lenStroke>
            <rateStroke uom="rpm">55</rateStroke>
            <pressure uom="psi">2100</pressure>
            <pcEfficiency uom="%">97</pcEfficiency>
            <pumpOutput uom="galUS/min">200</pumpOutput>
            <mdBit uom="m">7000</mdBit>
        </pumpOp>
        <shakerOp uid="so-1">
            <shaker uidRef="shaker-1">Shale shaker 1</shaker>
            <mdHole uom="m">7000</mdHole>
            <dTim>2001-10-25T06:00:00.000Z</dTim>
            <hoursRun uom="h">8</hoursRun>
            <pcScreenCovered uom="%">2</pcScreenCovered>
            <shakerScreen>
                <dTimStart>2001-10-25T06:00:00.000Z</dTimStart>
                <dTimEnd>2001-10-26T16:00:00.000Z</dTimEnd>
                <numDeck>1</numDeck>
                <meshX uom="mm">100</meshX>
                <meshY uom="mm">80</meshY>
                <manufacturer>Thule</manufacturer>
                <model>Th100</model>
                <cutPoint uom="mm">98</cutPoint>
            </shakerScreen>
        </shakerOp>
        <hse>
            <daysIncFree uom="d">0</daysIncFree>
            <incident uid="inc-1">
                <dTim>2001-11-01T08:15:00.000Z</dTim>
                <reporter>Joe Bloggs</reporter>
                <numMinorInjury>2</numMinorInjury>
```

```
                <numMajorInjury>2</numMajorInjury>
                <numFatality>1</numFatality>
                <isNearMiss>false</isNearMiss>
                <descLocation>Pipe deck</descLocation>
                <descAccident>Rough seas, casing strapped to pipe deck became loose
and rolled into working deck crew.</descAccident>
                <remedialActionDesc>Planned regular inspection of all restraining
straps, chains, etc.</remedialActionDesc>
                <causeDesc>Restraining chain snapped</causeDesc>
                <eTimLostGross uom="h">3000</eTimLostGross>
                <costLostGross currency="USD">3000000</costLostGross>
                <responsibleCompany>LowCost Chains Inc.</responsibleCompany>
            </incident>
            <lastCsgPresTest>2001-06-01T08:15:00.000Z</lastCsgPresTest>
            <presLastCsg uom="psi">5000</presLastCsg>
            <lastBopPresTest>2001-06-01T08:15:00.000Z</lastBopPresTest>
            <nextBopPresTest>2001-11-08T08:15:00.000Z</nextBopPresTest>
            <presStdPipe uom="psi">5000</presStdPipe>
            <presKellyHose uom="psi">5000</presKellyHose>
            <presDiverter uom="psi">5000</presDiverter>
            <presAnnular uom="psi">5000</presAnnular>
            <presRams uom="psi">5000</presRams>
            <presChokeLine uom="psi">5000</presChokeLine>
            <presChokeMan uom="psi">5000</presChokeMan>
            <lastFireBoatDrill>2001-06-01T08:15:00.000Z</lastFireBoatDrill>
            <lastAbandonDrill>2001-06-01T08:15:00.000Z</lastAbandonDrill>
            <lastRigInspection>2001-06-01T08:15:00.000Z</lastRigInspection>
            <lastSafetyMeeting>2001-06-01T08:15:00.000Z</lastSafetyMeeting>
            <lastSafetyInspection>2001-06-01T08:15:00.000Z</lastSafetyInspection>
            <lastTripDrill>2001-06-01T08:15:00.000Z</lastTripDrill>
            <lastDiverterDrill>2001-06-01T08:15:00.000Z</lastDiverterDrill>
            <lastBopDrill>2001-06-01T08:15:00.000Z</lastBopDrill>
            <regAgencyInsp>false</regAgencyInsp>
            <nonComplianceIssued>false</nonComplianceIssued>
            <numStopCards>1</numStopCards>
            <fluidDischarged uom="bbl">0</fluidDischarged>
            <volCtgDischarged uom="bbl">25</volCtgDischarged>
            <volOilCtgDischarge uom="bbl">1.5</volOilCtgDischarge>
            <wasteDischarged uom="bbl">0</wasteDischarged>
            <comments>No comments</comments>
        </hse>
        <personnel uid="per-1">
            <company>Exlog</company>
            <typeService>Mud log</typeService>
            <numPeople>4</numPeople>
            <totalTime uom="h">48</totalTime>
        </personnel>
        <personnel uid="per-2">
            <company>The Log Analyzers</company>
            <typeService>Analyze logs</typeService>
            <numPeople>2</numPeople>
            <totalTime uom="h">24</totalTime>
        </personnel>
        <supportCraft uid="boat-1">
            <name>Grampian-Provider</name>
            <typeSuppCraft>standby boat</typeSuppCraft>
            <dTimArrived>2001-11-01T08:15:00.000Z</dTimArrived>
            <dTimDeparted>2001-11-05T08:15:00.000Z</dTimDeparted>
            <comments>Onsite until 11/5/2001</comments>
        </supportCraft>
        <weather uid="w-1">
            <dTim>2001-11-01T08:15:00.000Z</dTim>
            <agency>Grampain Met Services</agency>
            <barometricPressure uom="mbar">811</barometricPressure>
            <beaufortScaleNumber>0</beaufortScaleNumber>
            <tempSurfaceMn uom="degC">4.9</tempSurfaceMn>
```

```
            <tempSurfaceMx uom="degC">6</tempSurfaceMx>
            <tempWindChill uom="degC">0</tempWindChill>
            <tempsea uom="degC">2</tempsea>
            <visibility uom="km">2</visibility>
            <aziWave uom="dega">16</aziWave>
            <htWave uom="m">4.4</htWave>
            <periodWave uom="s">6.9</periodWave>
            <aziWind uom="dega">16</aziWind>
            <velWind uom="m/s">11</velWind>
            <typePrecip>snow</typePrecip>
            <amtPrecip uom="in">1</amtPrecip>
            <coverCloud>Full</coverCloud>
            <ceilingCloud uom="ft">3000</ceilingCloud>
            <currentSea uom="m/h">1.4</currentSea>
            <aziCurrentSea uom="dega">100</aziCurrentSea>
            <comments>None</comments>
        </weather>
        <numAFE>FVNI/3000/07</numAFE>
        <costDay currency="USD">140535</costDay>
        <costDayMud currency="USD">2615</costDayMud>
        <diaHole uom="in">12.25</diaHole>
        <conditionHole>Good</conditionHole>
        <lithology>Sandstone</lithology>
        <nameFormation>Rotliegende</nameFormation>
        <diaCsgLast uom="in">13.375</diaCsgLast>
        <mdCsgLast uom="m">1797.4</mdCsgLast>
        <tvdCsgLast uom="m">1699</tvdCsgLast>
        <tvdLot uom="m">1703.4</tvdLot>
        <presLotEmw uom="g/L">1.55</presLotEmw>
        <presKickTol uom="psi">200</presKickTol>
        <volKickTol uom="bbl">65</volKickTol>
        <maasp uom="psi">3513</maasp>
        <tubular uidRef="45t12">TubAssy001</tubular>
        <sum24Hr>Drilled 12-1/4" hole to core point</sum24Hr>
        <forecast24Hr>Tag bottom with core assembly and cut core.</forecast24Hr>
        <statusCurrent>RIH with coring assembly.</statusCurrent>
        <commonData>
            <dTimCreation>2012-04-11T13:54:09.851Z</dTimCreation>
            <dTimLastChange>2012-04-12T07:14:05.071Z</dTimLastChange>
            <itemState>actual</itemState>
            <comments>Send 50bbl base oil back to base o/b Edda Fram</comments>
        </commonData>
    </opsReport>
</opsReports>
```

14.4 Example Output from Standard Query 12

This output is from an example for Standard Query 12 (SQ-012) found in Section 6.6.5.13, page 80. The example output is very long so it was moved to this appendix.

```
<mudLogs xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.witsml.org/schemas/1series" version="1.4.1.1"
xsi:schemaLocation="http://www.witsml.org/schemas/1series
http://www.witsml.org/schemas/1series/obj_mudLog.xsd">
      <!-- ================================================================== --
>
      <!-- Object=mudLog conversion from WITSML v1.3.1.1 to v1.4.1.1 requested. -->
      <!-- mode='write'. -->
      <!-- ================================================================== --
>
<mudLog uid="M-147155-1" uidWellbore="B-144490" uidWell="W-144490">
      <nameWell>DemoWell01</nameWell><nameWellbore>DemoWell01 - Main Wellbore
      </nameWellbore>
      <name>test - MudLog</name>
      <startMd uom="ft">5000</startMd>
      <endMd uom="ft">5060</endMd>
```

```
    <geologyInterval uid="MLGI-1003">
            <typeLithology>cuttings</typeLithology>
            <mdTop uom="ft">5000</mdTop>
            <mdBottom uom="ft">5030</mdBottom>
            <dTim>2001-10-31T08:15:00+08:00</dTim>
            <tvdTop uom="ft">4675</tvdTop>
            <tvdBase uom="ft">4680</tvdBase>
            <ropAv uom="ft/h">12</ropAv>
            <ropMn uom="ft/h">10</ropMn>
            <ropMx uom="ft/h">15</ropMx>
            <wobAv uom="klbf">0.006</wobAv>
            <tqAv uom="kft.lbf">0.6</tqAv>
            <rpmAv uom="rpm">120</rpmAv>
            <wtMudAv uom="lbm/galUS">1.403645913427</wtMudAv>
            <ecdTdAv uom="lbm/galUS">1.41701396974535</ecdTdAv>
            <dxcAv>0.95</dxcAv>
          <lithology uid="Lith001">
                <type>Sandstone</type>
                <codeLith>SS</codeLith>
                <lithPc uom="%">100</lithPc>
                <description>Sandstone: vf -f, clr-frost, mod srt, gd vis
            por</description>
                <lithClass>Lithology Classification</lithClass>
                <grainType>Grain type</grainType>
                <dunhamClass>Durham Class</dunhamClass>
                <color>Cir</color>
                <texture>Friable</texture>
                <hardness>Mod Hd</hardness>
                <sizeGrain>Fine</sizeGrain>
                <roundness>Mod Rnd</roundness>
                <sorting>Good</sorting>
                  <!--CHANGED: 'matrixCement' value-from 'Calcite'.'-->
                <matrixCement>calcite</matrixCement
                <porosityVisible>Mod</porosityVisible>
                <permeability>Permeability desc</permeability>
                <densShale uom="g/cm3">2.6</densShale>
            <qualifier uid="MLQ-001">
                      <type>Pyrite</type>
                      <mdTop uom="ft">5000</mdTop>
                      <mdBottom uom="ft">5030</mdBottom>
                      <abundance uom="%">10</abundance>
                      <abundanceCode>sparse</abundanceCode>
                      <description>lg crystals</description>
                </qualifier>
            </lithology>
          <show>
                <showRat>good</showRat>
                <stainColor>brown</stainColor>
                <stainDistr>even</stainDistr>
                <stainPc uom="%">50</stainPc>
                <natFlorColor>White</natFlorColor>
                <natFlorPc uom="%">25</natFlorPc>
                <natFlorLevel>faint</natFlorLevel>
                <natFlorDesc>Nat Flor Desc</natFlorDesc>
                <cutColor>white</cutColor>
                <cutSpeed>fast</cutSpeed>
                <cutStrength>strong</cutStrength>
                <cutForm>streaming</cutForm>
                <cutLevel>bright</cutLevel>
                <cutFlorColor>bright white</cutFlorColor>
                <cutFlorSpeed>instantaneous</cutFlorSpeed>
                <cutFlorStrength>strong</cutFlorStrength>
                <cutFlorForm>blooming</cutFlorForm>
                <cutFlorLevel>bright</cutFlorLevel>
                <residueColor>brown</residueColor>
                <showDesc>Show Desc</showDesc>
```

```
            <impregnatedLitho>Impregnated Litho</impregnatedLitho>
            <odor>petrol</odor>
        </show>
        <chromatograph>
            <dTim>2001-08-31T08:15:00+08:00</dTim>
            <mdTop uom="ft">5000</mdTop>
            <mdBottom uom="ft">5030</mdBottom>
            <wtMudIn uom="lbm/galUS">1.403645913427</wtMudIn>
            <wtMudOut uom="lbm/galUS">10.5</wtMudOut>
            <chromType>Chrom Type</chromType>
            <eTimChromCycle uom="s">30</eTimChromCycle>
            <chromIntRpt>2001-08-31T08:15:00+08:00</chromIntRpt>
            <methAv uom="ppm">600</methAv>
            <methMn uom="ppm">500</methMn>
            <methMx uom="ppm">700</methMx>
            <ethAv uom="ppm">120</ethAv>
            <ethMn uom="ppm">100</ethMn>
            <ethMx uom="ppm">130</ethMx>
            <propAv uom="ppm">45</propAv>
            <propMn uom="ppm">40</propMn>
            <propMx uom="ppm">70</propMx>
            <ibutAv uom="ppm">10</ibutAv>
            <ibutMn uom="ppm">5</ibutMn>
            <ibutMx uom="ppm">15</ibutMx>
            <nbutAv uom="ppm">10</nbutAv>
            <nbutMn uom="ppm">10</nbutMn>
            <nbutMx uom="ppm">10</nbutMx>
            <ipentAv uom="ppm">1</ipentAv>
            <ipentMn uom="ppm">1</ipentMn>
            <ipentMx uom="ppm">1</ipentMx>
            <npentAv uom="ppm">1</npentAv>
            <npentMn uom="ppm">0.099</npentMn>
            <npentMx uom="ppm">0.099</npentMx>
            <epentAv uom="ppm">0.099</epentAv>
            <epentMn uom="ppm">0.099</epentMn>
            <epentMx uom="ppm">0.099</epentMx>
            <ihexAv uom="ppm">0.099</ihexAv>
            <ihexMn uom="ppm">0.099</ihexMn>
            <ihexMx uom="ppm">0.099</ihexMx>
            <nhexAv uom="ppm">0.099</nhexAv>
            <nhexMn uom="ppm">0.099</nhexMn>
            <nhexMx uom="ppm">0.099</nhexMx>
            <co2Av uom="ppm">0.099</co2Av>
            <co2Mn uom="ppm">0.099</co2Mn>
            <co2Mx uom="ppm">0.099</co2Mx>
            <h2sAv uom="ppm">0</h2sAv>
            <h2sMn uom="ppm">0</h2sMn>
            <h2sMx uom="ppm">0</h2sMx>
            <acetylene uom="ppm">0.099</acetylene>
        </chromatograph>
        <mudGas>
            <gasAv uom="%">1.24</gasAv>
            <gasPeak uom="%">1.4</gasPeak>
            <gasPeakType>connection gas</gasPeakType>
            <gasBackgnd uom="%">1</gasBackgnd>
            <gasConAv uom="%">1.2</gasConAv>
            <gasConMx uom="%">1.3</gasConMx>
            <gasTrip uom="%">2.4</gasTrip>
        </mudGas>
            <densBulk uom="g/cm3">0.0249888028633778</densBulk>
            <densShale uom="g/cm3">0.0264304645670342</densShale>
            <calcite uom="%">6</calcite>
            <dolomite uom="%">1</dolomite>
            <cec uom="meq/g">0.00102604495942459</cec>
            <calcStab uom="%">0.099</calcStab>
            <nameFormation>Rotliegende</nameFormation>
```

```
      <lithostratigraphic>lithostratigraphic name</lithostratigraphic>
      <chronostratigraphic>Permian</chronostratigraphic>
      <sizeMn uom="in">0.099</sizeMn>
      <sizeMx uom="in">0.099</sizeMx>
      <lenPlug uom="in">0.099</lenPlug>
      <description>Sandstone: vf -f, clr-frost, mod srt, gd vis
por</description>
      <cuttingFluid>cuttingFluid desc</cuttingFluid>
      <cleaningMethod>dryingMethod desc</cleaningMethod>
      <dryingMethod>dryingMethod desc</dryingMethod>
      <comments>Geology Interval comments</comments>
 </geologyInterval>
<geologyInterval uid="MLGI-1004">
      <typeLithology>cuttings</typeLithology>
      <mdTop uom="ft">5030</mdTop>
      <mdBottom uom="ft">5060</mdBottom>
      <dTim>2001-10-31T08:15:10+08:00</dTim>
      <tvdTop uom="ft">4695</tvdTop>
      <tvdBase uom="ft">47000</tvdBase>
      <ropAv uom="ft/h">12</ropAv>
      <ropMn uom="ft/h">10</ropMn>
      <ropMx uom="ft/h">15</ropMx>
      <wobAv uom="klbf">0.006</wobAv>
      <tqAv uom="kft.lbf">0.6</tqAv>
      <rpmAv uom="rpm">120</rpmAv>
      <wtMudAv uom="lbm/galUS">1.403645913427</wtMudAv>
      <ecdTdAv uom="lbm/galUS">1.41701396974535</ecdTdAv>
      <dxcAv>0.95</dxcAv>
  <lithology uid="Lith001">
            <type>Sandstone</type>
            <codeLith>SS</codeLith>
            <lithPc uom="%">100</lithPc>
            <description>Sandstone: vf -f, clr-frost, mod srt, gd vis
      por</description>
            <lithClass>Lithology Classification</lithClass>
            <grainType>Grain type</grainType>
            <dunhamClass>Durham Class</dunhamClass>
            <color>Cir</color>
            <texture>Friable</texture>
            <hardness>Mod Hd</hardness>
            <sizeGrain>Fine</sizeGrain>
            <roundness>Mod Rnd</roundness>
            <sorting>Good</sorting>
               <!--CHANGED: 'matrixCement' value-from 'Calcite'.'-->
            <matrixCement>calcite</matrixCement>
            <porosityVisible>Mod</porosityVisible>
            <permeability>Permeability desc</permeability>
            <densShale uom="g/cm3">2.6</densShale>
       <qualifier uid="MLQ-001">
                  <type>Pyrite</type>
                  <mdTop uom="ft">5030</mdTop>
                  <mdBottom uom="ft">5060</mdBottom>
                  <abundance uom="%">10</abundance>
                  <abundanceCode>sparse</abundanceCode>
                  <description>lg crystals</description>
        </qualifier>
   </lithology>
      <show>
            <showRat>good</showRat>
            <stainColor>brown</stainColor>
            <stainDistr>even</stainDistr>
            <stainPc uom="%">50</stainPc>
            <natFlorColor>White</natFlorColor>
            <natFlorPc uom="%">25</natFlorPc>
            <natFlorLevel>faint</natFlorLevel>
            <natFlorDesc>Nat Flor Desc</natFlorDesc>
```

```
      <cutColor>white</cutColor>
      <cutSpeed>fast</cutSpeed>
      <cutStrength>strong</cutStrength>
      <cutForm>streaming</cutForm>
      <cutLevel>bright</cutLevel>
      <cutFlorColor>bright white</cutFlorColor>
      <cutFlorSpeed>instantaneous</cutFlorSpeed>
      <cutFlorStrength>strong</cutFlorStrength>
      <cutFlorForm>blooming</cutFlorForm>
      <cutFlorLevel>bright</cutFlorLevel>\<residueColor>brown</
residueColor>
      <showDesc>Show Desc</showDesc>
      <impregnatedLitho>Impregnated Litho</impregnatedLitho>
      <odor>petrol</odor>
   </show>
 <chromatograph>
      <dTim>2001-08-31T08:15:10+08:00</dTim>
      <mdTop uom="ft">5030</mdTop>
      <mdBottom uom="ft">5060</mdBottom>
      <wtMudIn uom="lbm/galUS">1.403645913427</wtMudIn>
      <wtMudOut uom="lbm/galUS">10.5</wtMudOut>
      <chromType>Chrom Type</chromType>
      <eTimChromCycle uom="s">30</eTimChromCycle>
      <chromIntRpt>2001-08-31T08:15:00+08:00</chromIntRpt>
      <methAv uom="ppm">600</methAv>
      <methMn uom="ppm">500</methMn>
      <methMx uom="ppm">700</methMx>
      <ethAv uom="ppm">120</ethAv>
      <ethMn uom="ppm">100</ethMn>
      <ethMx uom="ppm">130</ethMx>
      <propAv uom="ppm">45</propAv>
      <propMn uom="ppm">40</propMn>
      <propMx uom="ppm">70</propMx>
      <ibutAv uom="ppm">10</ibutAv>
      <ibutMn uom="ppm">5</ibutMn>
      <ibutMx uom="ppm">15</ibutMx>
      <nbutAv uom="ppm">10</nbutAv>
      <nbutMn uom="ppm">10</nbutMn>
      <nbutMx uom="ppm">10</nbutMx>
      <ipentAv uom="ppm">1</ipentAv>
      <ipentMn uom="ppm">1</ipentMn>
      <ipentMx uom="ppm">1</ipentMx>
      <npentAv uom="ppm">1</npentAv>
      <npentMn uom="ppm">0.099</npentMn>
      <npentMx uom="ppm">0.099</npentMx>
      <epentAv uom="ppm">0.099</epentAv>
      <epentMn uom="ppm">0.099</epentMn>
      <epentMx uom="ppm">0.099</epentMx>
      <ihexAv uom="ppm">0.099</ihexAv>
      <ihexMn uom="ppm">0.099</ihexMn>
      <ihexMx uom="ppm">0.099</ihexMx>
      <nhexAv uom="ppm">0.099</nhexAv>
      <nhexMn uom="ppm">0.099</nhexMn>
      <nhexMx uom="ppm">0.099</nhexMx>
      <co2Av uom="ppm">0.099</co2Av>
      <co2Mn uom="ppm">0.099</co2Mn>
      <co2Mx uom="ppm">0.099</co2Mx>
      <h2sAv uom="ppm">0</h2sAv>
      <h2sMn uom="ppm">0</h2sMn>
      <h2sMx uom="ppm">0</h2sMx>
      <acetylene uom="ppm">0.099</acetylene>
   </chromatograph>
 <mudGas>
      <gasAv uom="%">1.24</gasAv>
      <gasPeak uom="%">1.4</gasPeak>
      <gasPeakType>connection gas</gasPeakType>
```

```
                <gasBackgnd uom="%">1</gasBackgnd>
                <gasConAv uom="%">1.2</gasConAv>
                <gasConMx uom="%">1.3</gasConMx>
                <gasTrip uom="%">2.4</gasTrip>
            </mudGas>
              <densBulk uom="g/cm3">0.0249888028633778</densBulk>
              <densShale uom="g/cm3">0.0264304645670342</densShale>
              <calcite uom="%">6</calcite>
              <dolomite uom="%">1</dolomite>
              <cec uom="meq/g">0.00102604495942459</cec>
              <calcStab uom="%">0.099</calcStab>
              <nameFormation>Rotliegende</nameFormation>
              <lithostratigraphic>lithostratigraphic
            name</lithostratigraphic>
              <chronostratigraphic>Permian</chronostratigraphic>
              <sizeMn uom="in">0.099</sizeMn>
              <sizeMx uom="in">0.099</sizeMx>
              <lenPlug uom="in">0.099</lenPlug>
              <description>Sandstone: vf -f, clr-frost, mod srt, gd vis
            por</description>
              <cuttingFluid>cuttingFluid desc</cuttingFluid>
              <cleaningMethod>dryingMethod desc</cleaningMethod>
              <dryingMethod>dryingMethod desc</dryingMethod>
              <comments>Geology Interval comments</comments>
            </geologyInterval>
          <commonData>
                <sourceName>InterAct</sourceName>
                <dTimCreation>2012-04-16T08:57:29Z</dTimCreation>
                <dTimLastChange>2012-04-16T09:00:20Z</dTimLastChange>
            </commonData>
      </mudLog>
</mudLogs>
```

15. APPENDIX E: References

Organi-zations	Standards Designations	Standards Names	Dates	Remarks
IETF	HTTP/1.1	HTTP 1.1 Standard	June 1999	Used for transport.
W3C	SOAP 1.1	SOAP 1.1 Standard	8 May 2000	Used for transport.
W3C	WSDL 1.2	WSDL 1.2 Standard	3 March 2003	Used to define normative data structures of web service methods.
W3C	XML Schema 1.1	XML Schema Part 1: Structures	28 October 2004	Used to define data structures in WSDL and in Data Schemas.
W3C	XML Schema 1.1	XML Schema Part 2: Datatypes	28 October 2004	Used to define data structures in WSDL and in Data Schemas.
IETF	rfc 4122	A Universally Unique Identifier (UUID) URN Namespace	July 2005	May be used to create globally unique identifiers for independent data-objects.
W3C	HTML 4.01 Specification	application/x-www-form-urlencoded	24 December 1999	Used to encode parameter-value pairs.