WITSML v2.1 Release Notes

WITSML Overview	WITSML is the upstream oil and gas data-transfer standard for specifying and exchanging data for wells and well-related operations and objects, such as drilling, logging, and mud logging.
Version of Document	1.0
Date published	May 16, 2022
Prepared by	Energistics
Abstract	This document contains release notes for WITSML v2.1, which includes a list of changes from WITSML v2.0 to v2.1.
Document type	Release notes
Language	U.S. English

ENERGISTICS

ADOPT > ADVANCE > ACCELERATE

Table of Contents

1 WITSML Resources

These resources are included in the witsml folder of the energyML download package.

	Resource/Document	Description
1.	WITSML_v2.1_Release_Notes.pdf	The WITSML package includes release notes that summarize changes from WITSML v2.0 to v2.1 and provide some documentation of new features. See the Table of Contents.
2.	WITSML:XSD files	Schemas for all of the data objects defined in WITSML v2.1.
3.	WITSML UML Data Model (XMI file)	Energistics uses a UML data modeling tool to specify the data models for its domain standards and Energistics *common*. Scripts in the UML tool are then used to produce the set of XSD files (schemas). In many cases, UML diagrams are used in the documentation. Additionally, the UML data models are themselves included in the download package, in the relevant folders. Energistics saves the UML model as an XMI file, a format that can be imported by any UML data modeling tool.
4.	*WITSML Technical Reference Guide*	Lists and defines the schema folders, data objects, elements, and attributes, including related business rules. This document also identifies relationships between/among data objects and highlights significant information about relationships (if any). Generated from the witsml folder in the UML model.

For the list of resources for Energistics common (some of which must be used with WITSML) and the Energistics Common Technical Architecture, see *Download_Package_READ_ME.pdf*.

2 List of Changes from WITSML v2.0 to v2.1

This change list is organized according to the Jira tickets that were used to organize and do the work. Jira ticket WITSML-302 lists the high-level set of agreed priorities for WITSML v2.1 organized into these user stories, which has a corresponding section below that describes the change highlights:

- WITSML v1.4.1.1 parity (WITSML-303) (Section 2.1)
- Channel data object clean up (WITSML-304) (Section 2.2)
- Growing Data Object clean up (WITSML-306) (Section 2.3)
- JSON representation (WITSML-307) (Section 2.4)
- Other data object clean-up (WITSML-308) (Section 2.5)
- Datum/Trajectory/Tool Error Model enhancements (WITSML-326) (Section 2.6)
- Improvements to Energistics *common* (WITSM-358) (Section 2.7)
- OSDU Compatibility (WITSML-318) (Section 2.8)
- Documentation improvements (WITSML-305) (Section 2.9)

Each of these high-level Jira tickets has a detailed list of the related Jira tickets.

2.1 WITSML v1.4.1.1 Parity (WITSML-303)

The data objects listed below were in WITSML v1.4.1.1 but were not brought forward into WITSML v2.0 (because there were not subject matter experts (SMEs) to work on them).

- Target is now done and published.
- Conventional core (ConvCore) and sidewall core (SidewallCore) are NOT published in WITSML v2.1. However, they have been moved into WitsmlDesign package, and some initial work to convert them into the v2.0 design has been done. If in the future SMEs are available to work on the necessary design improvements, they should begin with these data objects in WitsmlDesign.

2.2 Channel data object clean up (WITSML-304)

For the detailed list of related JIRAs, see WITSML-304. Highlights include:

- 296: Introduced a lot of additional PWLS data and metadata. The Property Kind Dictionary still exists in Energistics *common*, and the full list of Channel kinds and Logging Tool Kinds from PWLS are now included as dictionaries in WITSML. For more information, see Chapter 4.
- Number of improvements to data types and indexes and metadata values that can be used for channels (e.g. channels indexed by pressure and temperature).
- Cleaned up channel design to better support the wireline use case, e.g., channels that have multiple passes of data, changes in direction, and still record all of the necessary metadata.
- Support for unitless channels.
- Reintroduced sensor offset so you can convey how far behind the drill bit or tool reference a sensor is.
- Simplified the design of the Channel and Channel Set data objects
- Complete review, clean-up, and addition (in some cases) of active status attribute, for data objects where it makes sense. Initially, WITSML had an active status on wellbore, log, trajectory and mud log. An active status has been added to other data objects, such as well, a rig, log channel set, etc.
- Improvements to how you can associate channels to the "things" that generated them. Renamed parent element to DataSource so you can point to many different objects (e.g., a rig, tubular object) indicating where the channel was produced.
- To better support data management workflows, Log and ChannelSets data objects can be created as "empty" (e.g., a ChannelSet with no channels) for example, when created initially in a data store (these objects are not expected to remain "empty").

- A Channel data object can now be delivered with data. (In v2.0, channel data could only be delivered in a ChannelSet). For more information and the associated rules, see Chapter 3.
- Cleaned up design and functionality related to secondary indexes on Channels and ChannelSets.

2.3 Growing Data Object clean up (WITSML-306)

Major effort to make the design consistent across all the kinds of growing data objects defined by WITSML. All parts from all objects tell you their range using the same element with same name, same documentation, etc.

- Removed the parts_ convention, which was previously used in v2.0 to send parts. Parts can now be sent using appropriate schemas in ETP.
- Active status information added in all relevant places (similarly as for Channel described above).
- Added index metadata on growing data objects, similarly as for Channel data object. EXAMPLE: You can create an empty trajectory or wellbore geology, you can advertise that MD will be in feet or meters, and other endpoints can be prepared to receive that data. This allows that receiving endpoint to establish business rules to help with data quality (e.g., that individual data points sent must have consistent UOM to the advertised metadata).

2.4 JSON representation (WITSML-307)

- The Energistics Architecture Team has defined how the JSON representation of WITSML (all ML) data objects should look but has not completed the *Energistics JSON Style Guide* nor defined JSON schemas. Sample data may be available on the Energistics Architecture Team Basecamp.
- Agreed the on-disk representation of data in the Channel and ChannelSet. For information, see Chapter 3.

2.5 Other Data Object Clean-up (WITSML-308)

- Design clean-up of the StimJob data object.
- Fixed/made consistent usages of data types in the schemas (e.g., uses of float to doubles and int to long).
- Fixed many things related to data object references (DORs). EXAMPLE: Many properties on data objects were strings or used an old format. These have all been cleaned up/corrected.
  - DOR has been redesigned and a new data object component reference (DataObjectComponentReference, DOCR) has been added. The DOCR lets you reference one or more particular components in data object. For more information, see Section 2.7.
- Added some missing IADC codes to bit information and tubular.
- Added something to tubular to clean up things and for OSDU compatibility.
- Added reference to BHA run in drill activity.
- Cleaned up how trajectories are included in reporting objects. OpsReport and DrillReport now include metadata (e.g., is azimuth reference to true north or grid north?), which is helpful in understanding the trajectory station data values included in the report.

2.6 Datum/Trajectory/Tool Error Model enhancements (WITSML-326)

- Significant redesign to trajectory, spatial location (datums, CRS, etc.; see Section 2.7) and tool error model.
- Well datum has now been replaced with reference points or reference points in a wellbore (depending on the specific use case). Those reference points are now data objects that can be shared and reused by multiple data objects and they reference back to the CRS that defines them.
  - Use of datum/CRS is now required in more places in WITSML, but it's easier because they can just be referenced.

2.7 Improvements to Energistics common (WITSM-358)

The changes here were made in Energistics common and are available for use in WITSML.

- Changes to Energistics identifiers, changes to DataObjectReference (DOR) and addition of DataObjectComponentReference (DOCR); for information, see the *Energistics Identifier Specification v5.0*.
- New BusinessAssociate data object to represent individuals, companies or other organizations and specify relevant data about an associate; this maps to the OSDU Organization Information. (NOTE: The BusinessAssociate business object was originally defined in PRODML but has been promoted to Energistics *common*.) In any data object, use a DOR to reference a BusinessAssociate. (Previously, in WITSML, there were data fields (strings) for this type of information (operator, manufacturer), which have all been removed now.)
- The Attachment data object has been removed from WITSML and promoted to Energistics common (so that it may be used by all the MLs). It has been redesigned so it may be attached to any data object with a DOR.
- Spatial location (CRS and datums) has been completely redesigned and moved to Energistics *common*. For more information, see the *Energistics Common Technical Architecture Overview Guide v2.3*.

2.8 OSDU Compatibility (WITSML-318)

This work was a detailed review of WITSML to ensure support of the data objects in OSDU (well, wellbore, well log, wellbore trajectory, wellbore marker set and tubular assembly (which includes tubular component and tubular umbilical).

You can move everything for OSDU (objects listed in previous paragraph) into WITSML and populate all data elements/attributes. You can also populate OSDU master data objects and work product components from the corresponding WITSML data objects.

Tubular components may not map as well as other data objects because OSDU and WITSML have taken fundamentally different design approaches to tubular. WITSML has many things that represents a tubular object (e.g., wellbore geometry that represents casing installed in the hole, tubular that represents a tool string or drill string, in the Completions part of the data model downhole component represents installed equipment inside the hole, etc.) In OSDU, all of these WITSML data objects map to a single OSDU data object, tubular assembly. WITSML has mapped the OSDU tubular assembly to the WITSML tubular data object only. Some of the OSDU elements are relevant for some of those other WITSML data objects, so groups of properties were created to accommodate those extra elements (e.g., they have been added for OSDU compatibility but don't add a lot of the WITSML design).

- Some elements attributes added in WITSML and some in Energistics *common*.
- Where it made sense, the WITSML data object design was modified. In some cases, simply added the necessary elements for OSDU integration.
- Added new PPFG channel, channel set and log data objects (see PPFG package) to carry information from the OSDU PPFG data set data object.

2.9 Documentation improvements (WITSML-305)

- Defined all the elements on data objects that should be immutable and managed by a server/store. This information is on the individual elements in the *WITSML Technical Reference Guide v2.1* and listed in Chapter 5.

3 On-disk Channel Data Representation

WITSML 2.1 has been designed to support a new JSON format for representing channel and channel set data on disk.

The design objectives were:

1. To be used when there is a need to store a single object (Log, ChannelSet or Channel) with both metadata and data.
2. Use a common JSON format that works in both XML and JSON representation of the data objects, for both individual channels and for channel sets.
3. Keep the format simple for the most common use cases while still capturing all the capabilities of the new channel format, such as multiple indices, point metadata, etc.

3.1 Format

The following format was designed to achieve these objectives.

1. Group channel data into an array of rows, one row per primary index value.
2. Every row must have the same representation, with the same number of elements in the array.
3. Each row is represented by a JSON Array.
   a. The first elements in the array, are the Indexes, in the same order as the Index metadata.
   b. Subsequent elements in the array is the data, in the same order as the channel metadata.

The JSON block may be stored in the ChannelSet or in the Channel in the 'Data' element.

The JSON should be enclosed in a CDATA block.

```
<xs:element name="Data" type="witsml:ChannelData" minOccurs="0"
maxOccurs="1"/>
```

RULES:

- When channels are delivered inside a ChannelSet data object, they must NOT have any content in the 'Data' element. The content may only occur in the ChannelSet.
- When Channel data objects are used independently, they may have content in the 'Data' element.

4. Data is represented using a single JSON object with two properties: metadata and data.
   a. The following are valid fields for metadata:
   - indexName
   - indexKind
   - indexUoM

   - channelUuid
   - channelName
   - channelDataType
   - channelUoM
   - channelAxisVectorLength
   - channelAttributeMetadata

   b. Any additional fields that are unknown to the reader must be ignored.

5. Data property contains an array of 'rows', where each row is an array. The first elements in the array are the indexes, followed by the data for the channels. The order of indexes and channels must match the metadata.

3.2 Data Representation

1. Data is represented as native JSON types.
2. Time is represented as a string in the extended profile of the ISO 8601-1:2019 format.
   a. Requires 'Z' for UTC time in the timezone field.
   b. Maximum useful number of fractional-seconds is 6 (microsecond precision) as this is the max precision of an ETP/WITSML time. Additional numbers are permitted as per the specification.
   c. A useful reference to this format can be found here: https://docs.microsoft.com/en-us/dotnet/standard/datetime/system-text-json-support#the-extended-iso-8601-12019-profile-in-systemtextjson
3. When a channel or secondary index does not have a data value at a given index, it's value must be null.
4. NaN, +infinity, -Infinity in JSON"
   a. For NaN it uses null. null is a valid entry for a numeric value, but it is impossible to distinguish between a value that is missing and a value that is present but not a valid number.
   b. To represent the numbers -Infinity and +Infinity, it uses the largest and smallest possible IEEE 754:2008 normal 64-bit numbers. Some environments already recognize these, but where they could be consumed or produced by an Energistics API, it must support them. The full number in scientific notation is (plus or minus) 1.7976931348623157E+308
5. Arrays (Vectors) are stored in the data as a 1D JSON array. If the array has more than one dimension, it must be flattened to a 1D array. The actual dimensions of the array are specified in the channelAxisVectorLength field.
6. When a data point has attributes (Point Metadata), the value and its attributes are stored as an array. The data point value is the first value of the array, and the point metadata values are the subsequent entries in the array.
   a. If a data point may have attributes, the channel's data points must always be represented as an array with elements for all possible attributes. Missing attribute values for a specific data point must be null.

The data requirements are slightly different between Channel and ChannelSet.

1. For metadata, in Channel there is a single entry in channel arrays; for ChannelSet there are multiple entries.
2. For data block, there is only one datapoint entry in the array per row (after the indexes).

3.2.1 Examples

CHANNEL Example (With one TIME index)

```
{
    "metadata": {
        "indexName": ["TIME" ],
        "indexKind": ["date time"],
        "indexUom": [null],
        "channelUuid": ["6638e572-89ca-4f5c-a698-47b538306a7a"],
        "channelName": ["GR"],
        "channelDataType": ["double"],
        "channelUom": ["gAPI"]
    },
    "data": [
        ["2009-06-22T05:21:03.0000000Z", 70.2],
        ["2009-06-22T05:21:04.0000000Z", 73.1],
        ["2009-06-22T05:21:05.0000000Z", 68.7],
        ["2009-06-22T05:21:06.0000000Z", 68.2],
        ["2009-06-22T05:21:07.0000000Z", 68.5],
        ["2009-06-22T05:21:08.0000000Z", 69.4],
        ["2009-06-22T05:21:09.0000000Z", 70.1],
        ["2009-06-22T05:21:10.0000000Z", 70.9],
        ["2009-06-22T05:21:11.0000000Z", 70.5]
    ]
}
```

CHANNEL Example (With one DEPTH index)

```
{
    "metadata": {
        "indexName": ["DEPTH" ],
        "indexKind": ["depth"],
        "indexUom": ["ft"],
        "channelUuid": ["6638e572-89ca-4f5c-a698-47b538306a7a"],
        "channelName": ["GR"],
        "channelDataType": ["double"],
        "channelUom": ["gAPI"]
    },
    "data": [
        [6253.0, 70.2],
        [6254.0, 73.1],
        [6255.0, 68.7],
        [6256.0, 68.6],
        [6257.0, 68.5],
        [6258.0, 69.4],
        [6259.0, 70.1],
        [6260.0, 70.9],
        [6261.0, 70.5]
    ]
}
```

CHANNEL Example (With two indexes)

```
{
    "metadata": {
        "indexName": ["TIME", "DEPTH"],
        "indexKind": ["date time", "depth"],
        "indexUom": [null, "ft"],
        "channelUuid": ["6638e572-89ca-4f5c-a698-47b538306a7a"],
        "channelName": ["GR"],
        "channelDataType": ["double"],
        "channelUom": ["gAPI"]
    },
    "data": [
        ["2009-06-22T05:21:03.0000000Z", 6253.0, 70.2],
        ["2009-06-22T05:21:04.0000000Z", 6252.0, 73.1],
        ["2009-06-22T05:21:05.0000000Z", 6353.0, 68.7],
        ["2009-06-22T05:21:06.0000000Z", 6251.0, 68.2],
        ["2009-06-22T05:21:07.0000000Z", 6251.0, null],
        ["2009-06-22T05:21:08.0000000Z", 6252.0, 69.4],
        ["2009-06-22T05:21:09.0000000Z", 6253.0, 70.1],
        ["2009-06-22T05:21:10.0000000Z", 6252.0, 70.9],
        ["2009-06-22T05:21:11.0000000Z", 6251.0, 70.5]
    ]
}
```

CHANNELSET Example (with one Index and 2 curves)

```
{
    "metadata": {
        "indexName": ["TIME", ],
        "indexKind": ["date time"],
        "indexUom": [null],
        "channelUuid": ["6638e572-89ca-4f5c-a698-
47b538306a7a","742cb419-3d78-4340-89c3-c6895d2646dc"],
        "channelName": ["ROPA","WOB"],
        "channelDataType": ["double"],
        "channelUom": ["ft/hr","Kg"]
    },
    "data": [
        ["2009-06-22T05:21:03.0000000Z", 70.2,1499],
        ["2009-06-22T05:21:04.0000000Z", 73.1,1500],
        ["2009-06-22T05:21:05.0000000Z", 68.7,1501],
        ["2009-06-22T05:21:06.0000000Z", 68.2,1498],
        ["2009-06-22T05:21:07.0000000Z", 68.5,null],
        ["2009-06-22T05:21:08.0000000Z", 69.4,1499],
        ["2009-06-22T05:21:09.0000000Z", 70.1,1500],
        ["2009-06-22T05:21:10.0000000Z", 70.9,1498],
        ["2009-06-22T05:21:11.0000000Z", 70.5,1505]
    ]
}
```

CHANNELSET: More complex example with Vectors and Point Metadata

```
{
    "metadata": {
        "indexName": ["TIME", "DEPTH"],
        "indexKind": ["date time", "measured depth"],
        "indexUom": [null, "ft"],
        "channelUuid": ["6638e572-89ca-4f5c-a698-47b538306a7a",
"04e0cab6-2663-44d6-b833-9ee4118b75bf"],
        "channelName": ["HKLD", "ARRAY"],
        "channelDataType": ["double", "string"],
        "channelUom": ["10 kN", null],
        "channelAxisVectorLength": [null, [3]],
        "channelAttributeMetadata": [null, [{"name": "TestAttribute",
"dataType": "boolean"}]]
    },
    "data": [
        ["2009-06-22T05:21:03.0000000Z", 6253.0, 140.3, [["a", "b",
"c"], false]],
        ["2009-06-22T05:21:04.0000000Z", 6253.0, 140.1, [["d", "e",
"f"], true]],
        ["2009-06-22T05:21:05.0000000Z", 6253.1, 142.7, null],
        ["2009-06-22T05:21:06.0000000Z", 6253.1, null, [["j", "k", "l"],
true]],
        ["2009-06-22T05:21:07.0000000Z", 6253.2, 141.8, [["m", "n",
"o"], false]]
    ]
}
```

4 PWLS: Updated and New Functionality

WITSML 2.1 has been designed, in part, to take better advantage of data from the latest version of the Practical Well Log Standard (PWLS).

NOTE: Data from PWLS is implemented as Energistics PropertyKinds in Energistics *common*; for more information, see the *Energistics Common Technical Architecture Overview Guide v2.3*. Also, the complete PWLS data set can be downloaded separately from the Energistics website at https://www.energistics.org/download-standards/).

The design objectives were:

- Preserve existing support for the PWLS Property Hierarchy.
- Add new support for PWLS tool and curve data.

To achieve these objectives, the following changes were made:

1. The PWLS PropertyKindDictionary was refreshed to reflect updates in the Energistics Common 2.3 data models as well as some minor updates to the dictionary data that were delivered with PWLS v3. The dictionary is located in the energyML download package here:

   common/v2.3/ancillary/PropertyKindDictionary_v2.3.xml

2. Introduce a new WITSML schema package: PWLS.xsd. This package includes the following new and revised enumerations and types:

   a. LoggingMethod. This enumeration was moved from the Log.xsd package to the PWLS package and expanded to include two additional values used in PWLS: "coiled tubing" and "subsea".

   b. LoggingToolClass. This is a new enumeration created to hold the PWLS "Well Log Tool Class" information.

   c. ChannelKind. This is a new type that holds information about a PWLS "Curve". A ChannelKind represents the common information about a kind of channel that is generated by a logging company. The common information includes the PropertyKind and standard mnemonic. A ChannelKind also references LoggingToolKind objects representing the logging tools from the logging company that generate channels of the kind. For example, Baker Hughes Inteq generates gamma ray channels with a standard mnemonic of GR from its DSL and GR tools.

   d. LoggingToolKind. This is a new type that holds information about a PWLS "Tool". A LoggingToolKind represents the common information about a kind of logging tool used by a logging company. The common information includes its name and unique identifier. It may also include its LoggingToolClass.

   e. ChannelKindDictionary and LoggingToolKindDictionary. These are new types that hold collections, respectively, of ChannelKind and LoggingToolKind data objects.

3. Include new ChannelKindDictionary and LoggingToolKind dictionaries that are populated with PWLS v3 data. The dictionaries are located in the energyML download package here:

   witsml/v2.1/ancillary/ChannelKindDictionary_v2.1.xml

   witsml/v2.1/ancillary/LoggingToolKindDictionary_v2.1.xml

4. Include optional references to ChannelKind and LoggingToolKind in the Log, ChannelSet and Channel data objects. This provides implementers new,standardized ways to store and query metadata about channel data.

5. Include additional, optional references to PropertyKind on ChannelIndex and PointMetadata. This provides implementers to store more explicit details about the kind of index or point metadata associated with their channel data.

Together, these features support a variety of new workflows:

- Logging companies can use ChannelKind and LoggingToolKind to deliver up-to-cate metadata about the kinds of channels they generate and tools they use alongside the channel data they deliver.
- Data consumers can find channels that have a certain PropertyKind. For example, they can find all types of gamma ray PropertyKinds and then find all Channels within a Wellbore with a gamma ray PropertyKind.
- Data managers can maintain a library of ChannelKind and LoggingToolKind data objects to track the kinds of tools and channels they have received data for from logging companies.
- A library of expected ChannelKind data objects can be used by a data consumer both to define expected channel metadata for data providers and as a convenient way to validate that received data matches expectations.
- LoggingToolKind data objects can be used to help organize channels of data into ChannelSets based on the tool that generated them.
- By inspecting the Channel and ChannelKind data objects with a certain PropertyKind, data consumers can identify all the different mnemonics that they have received that kind of data with, and they van validate whether or not those mnemonics match the standard mnemonic provided by the logging company. This can also be used to help standardize the mnemonics based on a library of "Global Mnemonics".

5 Immutable and Store-Managed Fields

Immutable elements on a data object may ONLY be set when an object is created. Thereafter, they MUST NOT be changed. If there is a need to change them, the object MUST be deleted and recreated.

Immutable elements on a growing object part may ONLY be set when the part is created. Thereafter, they MUST NOT be changed. If there is a need to change them, the part MUST be deleted and recreated.

NOTE: Previously, the Md, Tvd, MdBit and TvdBit elements on Wellbore had special behavior documented in the WITSML 1.4.1.1 API spec. In WITSML 2.x + ETP, they no longer have any special behavior, so are NOT included in the list of immutable or store managed fields.

5.1 Immutable Fields

All Objects

- uuid

Channel

- AxisDefinition
- AxisVectorLengths
- Mnemonic
- Unit
- DataKind
- DatumReference
- Index:
  - IndexKind, IndexPropertyKind, Uom, Direction, Mnemonic, DatumReference
- PointMetadata
  - Name, DataKind, Uom, MetadataPropertyKind, AxisDefinition, DatumReference

ChannelSet

- Index:
  - IndexKind, IndexPropertyKind, Uom, Direction, Mnemonic, DatumReference

WellboreMarkerSet

- MdInterval (and all subelements)

Growing Data Objects

Relevant Objects: CuttingsGeology, InterpretedGeology, ShowEvaluation, WellboreGeometry, Trajectory, MudLogReport

Index:

- IndexKind, IndexPropertyKind, Uom, Direction, DatumReference

5.2 Store Managed Fields

Channel

- PrimaryIndexInterval
- Data
- Index/IndexInterval

ChannelSet

PrimaryIndexInterval

Index/IndexInterval

Data

Log

- PrimaryIndexInterval

WellboreMarker

- Md (and all subelements)

Active Objects

- Relevant Objects: Channel, ChannelSet, Log, CuttingsGeology, InterpretedGeology, ShowEvaluation, WellboreGeology, WellboreGeometry, Trajectory, MudLogReport, Rig, Wellbore, Well
- ActiveStatus

Growing Data Objects

- Relevant Objects: CuttingsGeology, InterpretedGeology, ShowEvaluation, WellboreGeometry, Trajectory, MudLogReport
- MdInterval