MDIS OPC UA

Companion Specification

Release 1.2

October 3, 2018

Specification Type:	Industry Standard Specification	Comments:	
Title:	MDIS OPC UA Companion Specification	Date:	October 3rd, 2018
Version:	Release 1.2	Software:	MS-Word
		Source:	MDIS Companion Specification Release V1.2.doc
Author:	MDIS	Status:	Release

MDIS

UNIFIED ARCHITECTURE

FOREWORD

This specification is the specification for developers of OPC UA applications. The specification is a result of an analysis and design process completed by the MDIS member organisations to develop a standard interface to facilitate the development of applications by multiple vendors that shall inter-operate seamlessly together.

ISSUE REPORTING

The OPC Foundation strives to maintain the highest quality standards for its published specifications, hence they undergo constant review and refinement. Readers are encouraged to report any issues and view any existing errata here: http://www.opcfoundation.org/errata

Revision history

Rev.	Date	Description	Prepared by	Checked by	Approved By
1.0	01/13/2017	Initial release	P. Hunkar		
1.01	08/08/2017	Maintenance release	P. Hunkar		
1.2	10/22/2017	Updates for additional objects	P. Hunkar	MDIS	MDIS

Revision summary

Rev.	Affected sections	Description of change
1.0	All	Initial release
1.01	All	Maintenance release, no functional changes, just clarification resulting for test case definitions, inconsistence clarification and minor editorial fixes. See mantis description for detail on all updates.
1.2	Multiple	Added definitions for new Objects, including MDISTimeSync Object (new section 5.1.11, 5.9), MDISBaseInformation Object (new section 5.1.12, 5.10,6.2, 7.2.1), Signature transfer additions (Additions to valve definition in section 5.7, new reference type section 8.3, general AddressSpace information (section 9.2). Profiles were updated to include new profiles for new functionality (section 12)

Revision 1.2 Highlights
The following table includes the Mantis issues resolved with this revision.

Mantis ID	Summary	Resolution
3937	Sync and read Time with MDIS	Added functionality to specification
3843	Valve signature/profile retrieval	Added functionality to allow a client to retrieve signatures from a server

Table Of Contents

List of figures

List of tables

1 MDIS

1.1 Introduction

This document defines the OPC UA MDIS companion specification. This standard is an Oil and Gas standard for interfacing the Subsea Production Control System (SPCS), with a Master Control Station (MCS) or a Subsea Gateway, to the Distributed Control System (DCS). This specification includes:

- A description of common terms,
- Supported architectures,
- Information models representing the data that is shared between the systems,
- *Methods* used in the flow of information,
- *Profile* & *ConformanceUnits* describing the grouping of functionality,
- Recommended Practices for the use of MDIS.

MDIS was created to define a standard object model that is transported on a common high performance efficient interface between topside and subsea systems in Oil and Gas installations. The selected protocol (OPC UA) includes independent third-party certification.

1.2 Concepts / Definitions

1.2.1 Introduction

This document makes use of a number of terms and concepts that are described in this section. OPC defined terms or terms defined in this document are in italics and camel case.

1.2.2 API Standard 17F concepts

DCS (Distributed Control System) is the production facility control system provides a centralised control system for the facility for the operators and is used to monitor and control the subsea production system. The DCS system will normally host the OPC UA *Client*.

MCS (Master Control Station) is the central control node containing application software required to control and monitor the subsea production system.

Subsea Gateway provides a communications interface on the surface to the subsea control equipment over the subsea vendor’s communication system. The Subsea Gateway may form part of the overall MCS. The MCS or Subsea Gateway will normally host the OPC UA *Server*.

SCV (Subsea Controls Vendor) equipment refers to the subsea vendor supplied equipment and principally includes the subsea gateway in “Integrated” architecture or the subsea gateway and MCS in “Interfaced” architecture (see section 4 for additional details). It is intended to classify the functionality that is delivered by the subsea vendor whether it is implemented in the MCS or subsea gateway.

Note: The MCS may be supplied by a vendor other than the subsea or DCS vendor.

HPU (Hydraulic Power Unit) is the unit which provides low pressure and high pressure hydraulic supplies for the control of subsea wells.

EPU (Electrical Power Unit) is the unit which provides power to the subsea system.

The Customer is the end user, typically the “Operating Company”.

The Operator is the human being executing an operation, not to be confused with the “Operating Company”.

1.2.3 MDIS Mandatory & Optional Items / Objects

MDIS has standardised on the following definitions for *Mandatory* and *Optional* items. In all cases (*Mandatory* or *Optional*), if the item is available, the functionality described by the definition of the item must be correct and verifiable.

Mandatory

Objects specified as *Mandatory* will be required in all *Objects* and cannot be deleted. In OPC terms, a *Mandatory* item must exist on every *Node* of the *NodeClass*, for example if an *MDISValveObjectType* defines a *Mandatory* item *Position* then every instance of the *MDISValveObjectType* must have an item *Position* available.

Optional

Objects have functionality that may or may not be included; if they are included the OPC *Client* will know how to handle them. In OPC terms, an *Optional* item may or may not exist on every *Node* of the *NodeClass*, for example if an *MDISValveObjectType* defines an *Optional* item *OpenTimeDuration* then some instances of the *MDISValveObjectType* may contain an item *OpenTimeDuration*, but other instances may not. *Clients* are required to be able to handle the case where the item does not exist.

1.2.4 OPC Compliance & Certification

In regard to OPC Testing for Compliance and Certification the following concepts apply. The actual requirements for certification are defined on a project basis, but the standard third party certification provided by the OPC Foundation provides the following:

Compliance - Assurance that the OPC UA *Server* or *Client* fulfils all functionality that it claims to support in terms of *Profiles* and that of all exposed interfaces function as defined in the specifications.

Interoperability - Testing of products against other products, this includes all functionality, data types and access rights.

Robustness - The testing of failure cases including handling of lost communication, communication recovery. All problems must not affect other connections, quality information must be correctly reported, and audit entries are generated as needed. In short, end users are aware of any problem and problems are resolved automatically where possible.

Efficiency - The testing of products under load, forcing of noisy / bad communications and ensuring that products continue to work. Measuring CPU, RAM, threads, handles etc. and ensuring that even under the poor communications, heavily loaded *Servers* and *Clients* continue to function and not leak resources.

Usability - Verify that products are delivered with some level of documentation and that the documentation that is provided is accurate and understandable. Verify that the product functions as advertised and an end user would understand what is being provided.

Certification - Validation of *Server* or *Client* products. Certification includes compliance, interoperability, robustness, efficiency and usability testing and results in a seal of approval from an OPC Foundation test lab upon meeting or exceeding defined acceptance criteria.

1.3 OPC Definitions

Table 1 lists OPC UA definitions which are used in this document, they are included here as a reference. Additional information can be found in the reference documents listed in section 2.

Table 1 - OPC UA Terms and Definitions

Term	Definition
AddressSpace	The collection of information that an OPC UA *Server* makes visible to its *Clients*. See Part 3 – Address Space Model for a description of the contents and structure of the *Server AddressSpace*.
Attribute	A primitive characteristic of a *Node*. All *Attributes* are defined by OPC UA, and may not be defined by *Clients* or *Servers*. *Attributes* are the only elements in the *AddressSpace* permitted to have data values. See Part 3 – Address Space Model for additional details.

Term	Definition
ConformanceUnit	As defined by the OPC Foundation: a specific set of OPC UA features that can be tested as a single entity. As it applies to MDIS a *ConformanceUnit* may describe a specific *Object* or part of a specific *Object*. It may also describe general functionality such as redundancy or performance. For each *ConformanceUnit* one or more test cases will exist to ensure that the defined functionality is provided. The test cases may be automatically executed in a Compliance Test Tool (CTT) or they may require some level of manual interaction. See Part 7 – Profiles for additional details.
Facet	A *Profile* that describes a subset of functionality. This functionality must be paired with other *Facets* or *Profile*s to provide an operating *Server* or *Client*. See Part 7 – Profiles for additional details.
InformationModel	An organisational framework that defines, characterises and relates information resources of a given system or set of systems. The core address space model supports the representation of *InformationModels* in the *AddressSpace*. See Part 5 – Information Model for a description of the base OPC UA Information Model.
Method	A callable software function that is a component of an *Object*. See Part 4 – Services for a basic definition and see Part 10 – Programs for advanced uses.
Node	The fundamental component of an *AddressSpace*. See Part 3 – Address Space Model for additional details.
NodeClass	The class of a *Node* in an *AddressSpace*. *NodeClasses* define the metadata for the components of the OPC UA *Object* Model. They also define constructs, such as *Views*, that are used to organise the *AddressSpace*. See Part 3 – Address Space Model for additional details.
Object	Objects from an object-oriented technology point of view would have the following definition. Objects share two characteristics: They have state (*Attribute*) and behaviour (*Method*). A bicycle has states (current gear, current pedal cadence, current speed) and behaviour (changing gear, changing pedal cadence, applying brakes). An object stores its state in fields (*Variables*) and exposes its behaviour through functions (*Methods*). Functions operate on an object's internal state and serve as the primary mechanism for object-to-object communication. Hiding internal state and requiring all interaction to be performed through an object's functions is known as data encapsulation, a fundamental principle of object-oriented programming. In programming languages this object will have a third characteristic: The identity, which will help to find and use the object. From an OPC UA point of view the following definition is used: A *Node* that represents a physical or abstract element of a system. *Objects* are modelled using the OPC UA *Object* Model. Systems, subsystems and devices are examples of *Objects*. An *Object* is defined as an instance of an *ObjectType*. See Part 3 – Address Space Model for additional details.
Object Instance	A synonym for *Object*. See Part 3 – Address Space Model and Part 5 – Information Model for additional details.
ObjectType	A *Node* that represents the *TypeDefinition* for an *Object*. See Part 3 – Address Space Model and Part 5 – Information Model for additional details.

Term	Definition
Profile	A specific set of capabilities, to which a *Server* or *Client* may claim conformance. The capabilities are defined by a set of *ConformanceUnits*. Each *Server* or *Client* may claim conformance to more than one *Profile*. The OPC Foundation provides a base list of *Server* and *Client Profiles* and *Facets* in an online database which is also documented in an OPC UA specification, Part 7 – Profiles. The online database can be found here: (http://opcf.org/profilereporting/index.htm?All=true),
Property	A *Variable* that is a leaf and cannot have any children. See Part 3 – Address Space Model and Part 5 – Information Model for additional details.
Reference	An explicit relationship (a named pointer) from one *Node* to another. The *Node* that contains the *Reference* is the source *Node*, and the referenced *Node* is the target *Node*. All *References* are defined by *ReferenceTypes*. See Part 3 – Address Space Model and Part 5 – Information Model for additional details.
ReferenceType	A *Node* that represents the *TypeDefinition* of a *Reference*. The *ReferenceType* specifies the semantics of a *Reference*. The name of a *ReferenceType* identifies how source *Nodes* are related to *TargetNodes* and generally reflects an operation between the two, such as "A contains B". See Part 3 – Address Space Model and Part 5 – Information Model for additional details.
UANodeSet	The root of the *AddressSpace* defined in an XML document. It defines a set of *Nodes*, their *Attributes* and *References*. See Part 6 – Mappings for additional details.
Variable	*Variable* is a *Node* that contains a value and can have children. See Part 3 – Address Space Model and Part 5 – Information Model for additional details.
VariableType	A *Node* that represents the *TypeDefinition* for a *Variable*. See Part 3 – Address Space Model and Part 5 – Information Model for additional details.

1.4 Industry Typical Abbreviations, Acronyms & Definitions

The abbreviations, acronyms and definitions listed in Table 2 are typical and primarily focused on Subsea projects although some Topsides specific terms are also included.

Table 2 - Abbreviations, Acronyms and Definitions

Abbreviations, Acronyms & Definitions	
API	American Petroleum Institute
CIMV	Chemical Injection Metering Valves
CSV	Comma Separated Values
DCS	Distributed Control System
EPU	Electrical Power Unit (Part of PCU)
ERP	Enterprise Resource Planning
EU	Engineering Units
FEED	Front End Engineering Design
HMI	Human Machine Interface
HTTP	Hypertext Transfer Protocol
IEC	International Electrotechnical Commission
JIP	Joint Industry Project
LVDT	Linear Variable Displacement (Differential) Transmitter
MCS	Master Control Station (Subsea Process Control System)

Abbreviations, Acronyms & Definitions	
MDIS	MCS-DCS Interface Standardisation (Industry JIP)
MPFM	Multiphase Flow Meters
NTP	Network Time Protocol
OLE	Object Linking & Embedding
OPC	Open Process Control (original Classic was OLE for Process Control)
OPC UA	OPC Unified Architecture
ROV	Remotely Operated Vehicle
SCADA	Supervisory Control And Data Acquisition
SCV	Subsea Controls Vendor
SEM	Subsea Electronics Module
SIS	Safety Instrumented System
SPCS	Subsea Production Control System
TCP/IP	Transmission Control Protocol / Internet Protocol
UML	Unified Modelling Language
URL	Uniform Resource Locator
XML	Extensible Mark-up Language

2 Reference documents

The OPC UA Specifications are organised as a multi-part document. The following list of references provide link to each of the parts. Also referenced are applicable API documents. This companion specification assumes that all OPC UA Application support as a minimum version 1.02 of the OPC UA specifications.

Part 1: OPC UA Specification: Part 1 – Concepts
http://www.opcfoundation.org/UA/Part1/

Part 2: OPC UA Specification: Part 2 – Security Model
http://www.opcfoundation.org/UA/Part2/

Part 3: OPC UA Specification: Part 3 – Address Space Model
http://www.opcfoundation.org/UA/Part3/

Part 4: OPC UA Specification: Part 4 – Services
http://www.opcfoundation.org/UA/Part4/

Part 5: OPC UA Specification: Part 5 – Information Model
http://www.opcfoundation.org/UA/Part5/

Part 6: OPC UA Specification: Part 6 – Mappings
http://www.opcfoundation.org/UA/Part6/

Part 7: OPC UA Specification: Part 7 – Profiles
http://www.opcfoundation.org/UA/Part7/

Part 8: OPC UA Specification: Part 8 – Data Access
http://www.opcfoundation.org/UA/Part8/

Part 9: OPC UA Specification: Part 9 – Alarms and Conditions
http://www.opcfoundation.org/UA/Part9/

Part 10: OPC UA Specification: Part 10 – Programs
http://www.opcfoundation.org/UA/Part10/

Part 11: OPC UA Specification: Part 11 – Historical Access
http://www.opcfoundation.org/UA/Part11/

Part 12: OPC UA Specification: Part 12 – Discovery
http://www.opcfoundation.org/UA/Part12/

Part 13: OPC UA Specification: Part 13 - Aggregates
http://www.opcfoundation.org/UA/Part13/

API Standard 17F - Standard for Subsea Production Control Systems:
http://www.api.org/products-and-services/standards/purchase

The OPC UA Specifications are also available from the IEC as IEC 62541

3 OPC UA Overview

3.1 Introduction

For the MDIS user who may not be familiar with OPC UA, the following section provides a brief overview of key features. It does not describe how MDIS makes use of these features it only describes the features available in OPC UA. MDIS specific functionality is specified in other sections of this document.

3.2 What is OPC UA?

OPC UA is an open and royalty free standard designed as a universal communications protocol. It is also available as IEC 62541.

OPC UA has a broad scope which delivers economies of scale for application developers. When combined with powerful semantic models, OPC UA makes it easier for end users to access data via generic commercial application. It provides an information modelling framework that allows application developers to represent their data in a way that makes sense to them.

The OPC UA model is scalable from small devices to Enterprise Resource Planning (ERP) systems. OPC UA devices process information locally and then provides that data in a consistent format to any application requesting data. For a more complete overview see Part 1 – Concepts.

3.3 Basics of OPC UA

As an Open Standard, OPC UA is based on standard Internet technologies, such as TCP/IP, HTTP, Ethernet, and XML. OPC UA provides a set of services (see Part 4 – Services) and a basic information model framework.

As an Extensible Standard, OPC UA provides an information model framework which can expose vendor defined information in a standard way. More importantly all OPC UA *Clients* are expected to be able to discover and use vendor defined information. This means OPC UA users can benefit from the economies of scale that come with generic visualisation and interface applications. This specification is an example of an OPC UA *InformationModel* designed to meet the needs of developers and users in the offshore oil and gas industry.

OPC UA *Clients* can be any consumer of data, from devices / controllers on the network; browser based thin clients and higher level ERP systems. OPC UA applications are platform and development language dependant. The full scope of OPC UA applications are illustrated in Figure 1. For this companion specification the typical communication would be device to device or device to SCADA type communications.

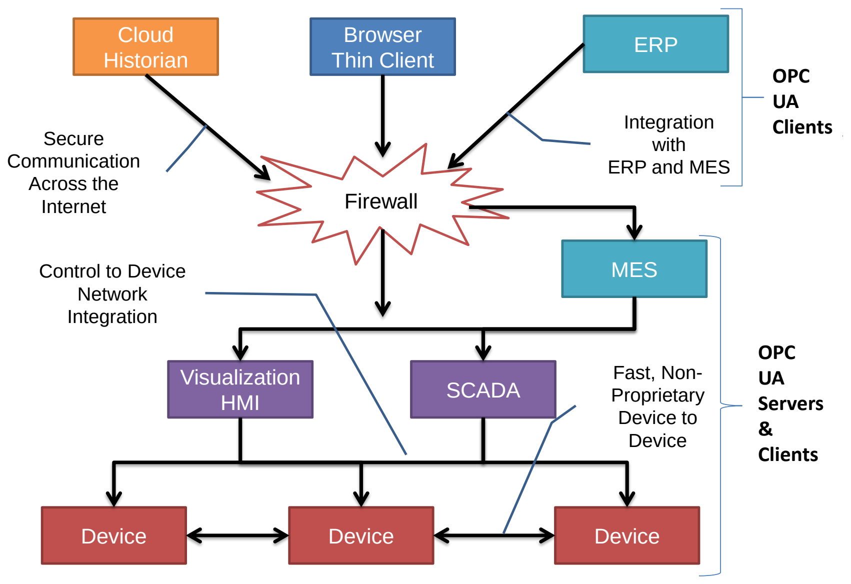
Cloud Historian
Browser Thin Client
ERP
OPC UA Clients
Secure Communication Across the Internet
Integration with ERP and MES
Firewall
MES
Control to Device Network Integration
Visualization HMI
SCADA
Fast, Non-Proprietary Device to Device
OPC UA Servers & Clients
Device
Device
Device

Figure 1 - The Scope of OPC UA within an Enterprise

OPC UA provides a robust and reliable communication infrastructure having mechanisms for handling lost messages, recovering from network interruptions, etc. With its binary encoded data it offers a high-performance data exchange solution. Security is built into OPC UA, security requirements are becoming more and more important as, increasingly, environments are connected to the office network or the internet and attackers are starting to focus on automation systems

3.4 Information Modelling in OPC UA

3.4.1 Concepts

OPC UA provides a framework that can be used to represent complex information as *Objects* in an *AddressSpace* which can be accessed with standard web services. These *Objects* consist of *Nodes* connected by *References*. Different classes of *Nodes* convey different semantics. For example, a *Variable Node* represents a value that can be read or written. The *Variable Node* has an associated *DataType* that can define the actual value, such as a string, float, structure etc. It can also describe the *Variable* value as a variant. A *Method Node* represents a function that can be called. Every *Node* has a number of *Attributes* including a unique identifier called a *NodeId* and non-localised name called a *BrowseName*. An *Object* representing a Heater is shown in Figure 2.

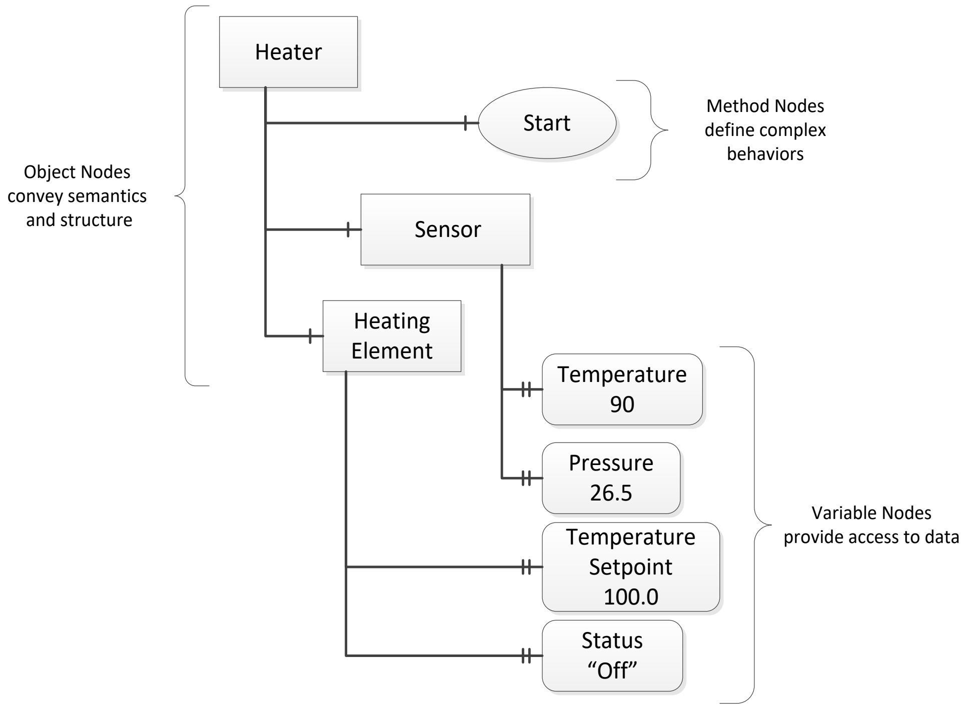
Heater
Start
Method Nodes define complex behaviors
Object Nodes convey semantics and structure
Sensor
Heating Element
Temperature 90
Pressure 26.5
Temperature Setpoint 100.0
Status "Off"
Variable Nodes provide access to data

Figure 2 - A Basic Object in an OPC UA Address Space

Object and *Variable Nodes* are called *Instance Nodes* and they always reference a *TypeDefinition* (*ObjectType* or *VariableType*) *Node* which describes their semantics and structure. Figure 3 illustrates the relationship between an instance and its *TypeDefinition*.

Type *Nodes* are templates that define all of the children that can be present in an instance of the type. In the example in Figure 3 the BoilerType *ObjectType* defines two sensors: Pressure and Temperature. All instances of BoilerType are expected to have the same children with the same *BrowseName*s. Within a type the *BrowseName*s uniquely identify the child. This means *Client* applications can be designed to search for children based on the *BrowseNames* from the type instead of *NodeIds*. This eliminates the need for manual reconfiguration of systems if a *Client* uses types that multiple devices implement.

OPC UA also supports the concept of subtyping. This allows a modeller to take an existing type and extend it. There are rules regarding subtyping defined in Part 3 – Address Space Model, but in general they allow the extension of a given type or the restriction of a *DataType*. For example, the modeller may decide that the existing *ObjectType* in some cases needs an additional *Variable*. The modeller can create a subtype of the Object and add the *Variable*. A *Client* that is expecting the parent type can treat the new type as if it was of the parent type. With regard to *DataTypes*, if a

Variable is defined to have a numeric value, a subtype could restrict the value to a float. This standard adds additional rules for extensions.

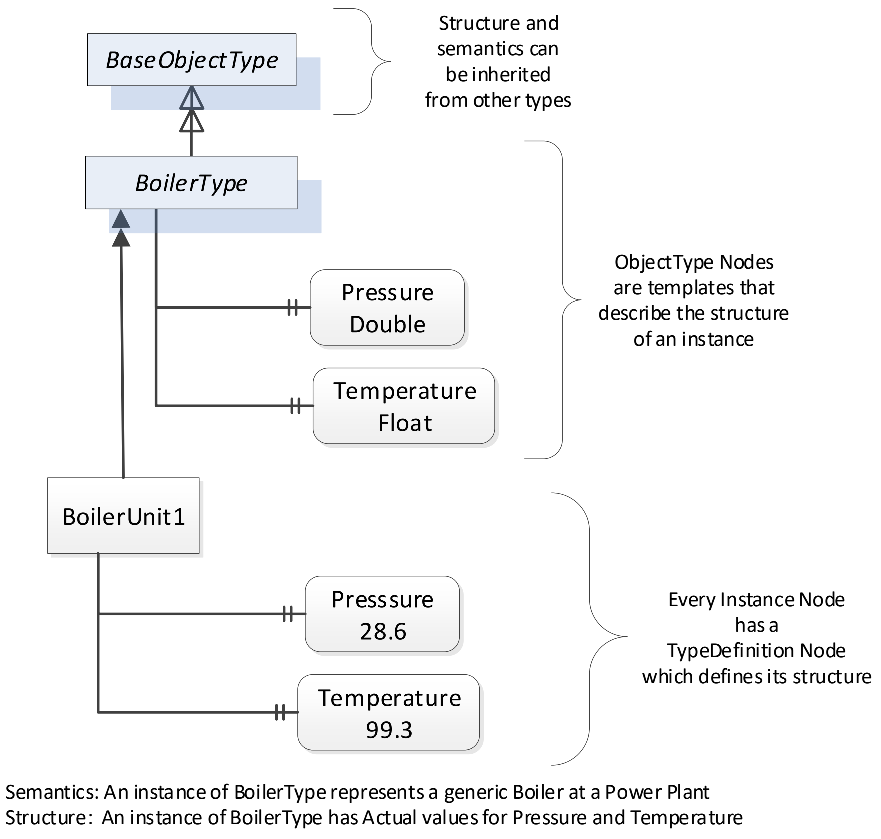
BaseObjectType
Structure and semantics can be inherited from other types
BoilerType
Pressure Double
Temperature Float
ObjectType Nodes are templates that describe the structure of an instance
BoilerUnit1
Presssure 28.6
Temperature 99.3
Every Instance Node has a TypeDefinition Node which defines its structure
Semantics: An instance of BoilerType represents a generic Boiler at a Power Plant
Structure: An instance of BoilerType has Actual values for Pressure and Temperature

Figure 3 - The Relationship between Type Definitions and Instances

References allow *Nodes* to be connected together in ways that describe their relationships. All *References* have a *ReferenceType* that specifies the semantics of the relationship. *References* can be hierarchical or non-hierarchical. Hierarchical *References* are used to create the structure of *Objects* and *Variables*. Non-hierarchical *References* are used to create arbitrary associations. Applications can define their own *ReferenceType* by creating subtypes of the existing *ReferenceType*. Subtypes inherit the semantics of the parent but may add additional restrictions. Figure 4 depicts several *References* connecting different *Objects*.

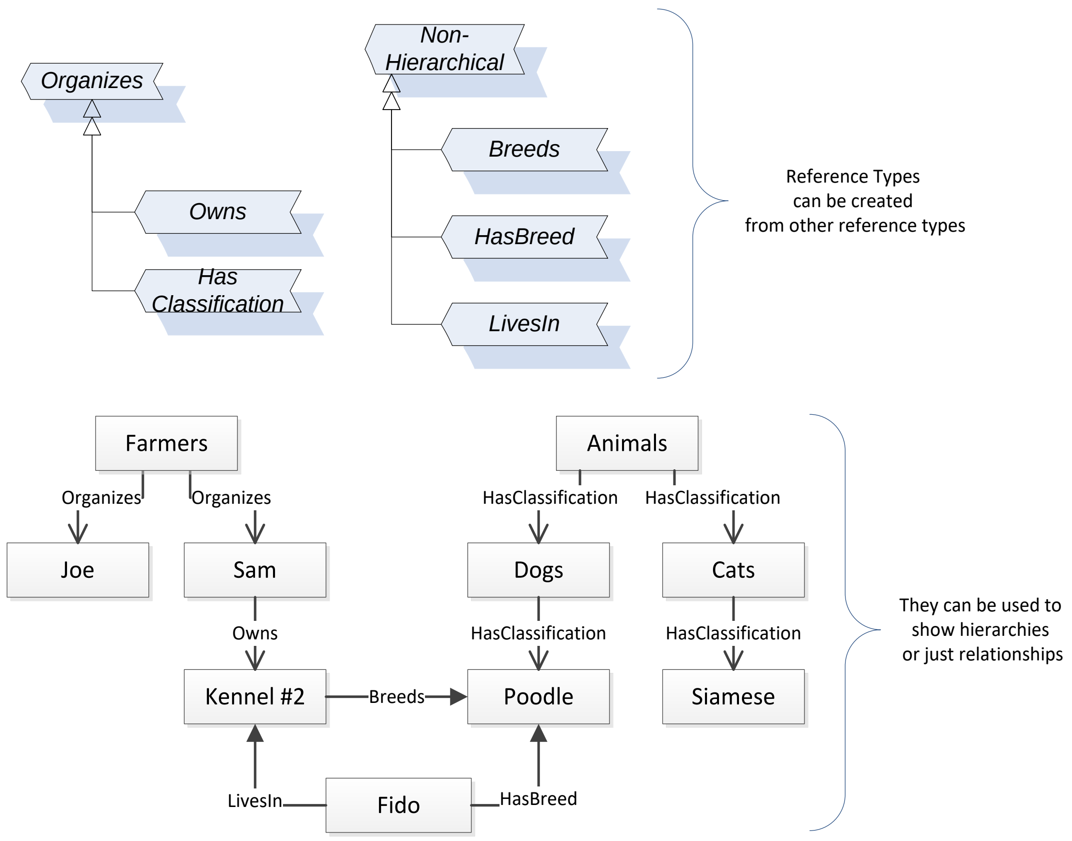

Figure 4 - Examples of References between Objects

The figures above use a notation that was developed for the OPC UA specification. The notation is summarised in Figure 5. UML representations can also be used; however, the OPC UA notation is less ambiguous because there is a direct mapping from the elements in the figures to *Nodes* in the *AddressSpace* of an OPC UA *Server*.

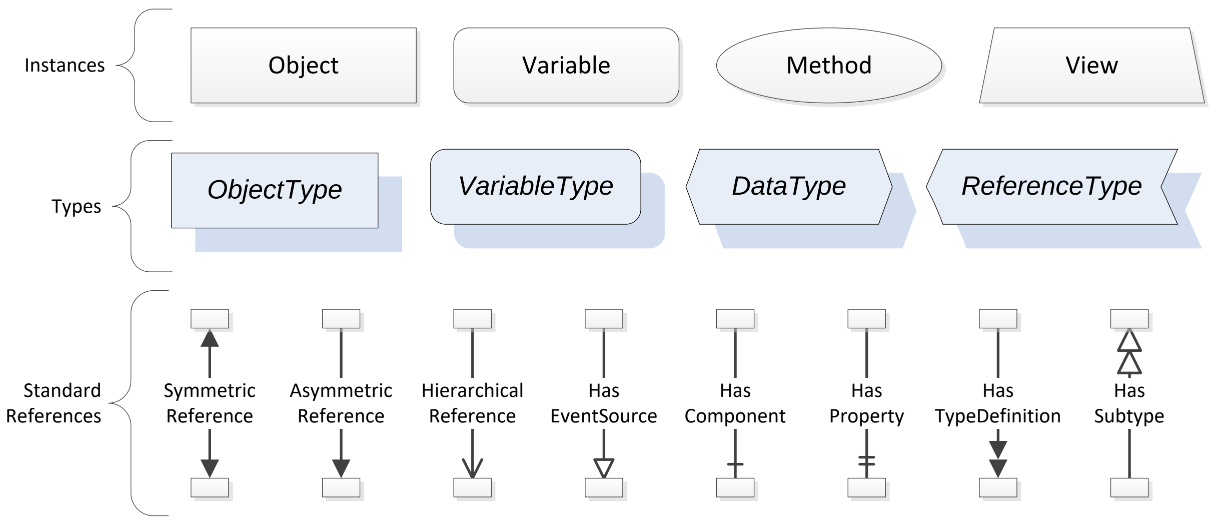

Figure 5 - The OPC UA Information Model Notation

A complete description of the different types of *Nodes* and *References* can be found in Part 3 – Address Space Model and the base OPC UA *AddressSpace* is described in Part 5 – Information Model.

The OPC UA specification defines a very wide range of functionality in its basic information model. It is not expected that all *Clients* or *Servers* support all functionality in the OPC UA specifications. OPC UA includes the concept of *Profiles*, which segment the functionality into testable certifiable units.

This allows the development of companion specifications (such as OPC UA MDIS) that can describe the subset of functionality that is expected to be implemented. The *Profiles* do not restrict functionality but generate requirements for a minimum set of functionality (see Part 7 – Profiles).

The OPC Foundation also defines a set of *InformationModels* that provide a basic set of functionalities. The Data Access specification (see Part 8 – Data Access) provides a basic *InformationModel* for typical process or measured data. The Alarm and Condition specification (see Part 9 – Alarms and Conditions) defines a standard *InformationModel* for *Alarms* and *Conditions*. The Programs specification (see Part 10 – Programs) defines a standard *InformationModel* for extending the functionality available via *Method* calls and state machines. The Historical Access specification (see Part 11 – Historical Access) defines the *InformationModel* associated with Historical Data and Historical Events. The aggregates specification (see Part 13 - Aggregates) defines a series of standard aggregate functions that allow a *Client* to request summary data. Examples of aggregates include averages, minimums, time in state, standard deviation, etc.

3.4.2 Namespaces

OPC UA allows information from many different sources to be combined into a single coherent *AddressSpace*. *Namespaces* are used to make this possible by eliminating naming and id conflicts between information from different sources. *Namespaces* in OPC UA have a globally unique string called a *NamespaceUri* and a locally unique integer called a *NamespaceIndex*. The *NamespaceIndex* is only unique within the context of a Session between an OPC UA *Client* and an OPC UA *Server*. All of the web services defined for OPC UA use the *NamespaceIndex* to specify the *Namespace* for qualified values.

There are two types of values in OPC UA that are qualified with *Namespaces*: *NodeIds* and *QualifiedNames*. *NodeIds* are globally unique identifiers for *Nodes*. This means the same *Node* with the same *NodeId* can appear in many *Servers*. This, in turn, means *Clients* can have built in knowledge of some *Nodes*. OPC UA *InformationModels* generally define globally unique *NodeIds* for the *TypeDefinitions* defined by the *InformationModel*.

QualifiedNames are non-localised names qualified with a *Namespace*. They are used for the *BrowseNames* of *Nodes* and allow the same names to be used by different *InformationModels* without conflict. The *BrowseName* is used to identify the children within a *TypeDefinition*. Instances of a *TypeDefinition* are expected to have children with the same *BrowseName*s. *TypeDefinitions* are not allowed to have children with duplicate *BrowseNames*; however, instances do not have that restriction.

3.4.3 Companion Specifications

An OPC UA companion specification for an industry specific vertical market describes an *InformationModel* by defining *ObjectTypes*, *VariableTypes*, *DataTypes* and *ReferenceTypes* that represent the concepts used in the vertical market. Table 3 contains an example of an *ObjectType* definition.

Table 3 - Example ObjectType Definition

Attribute	**Value**				
BrowseName	BoilerType				
IsAbstract	False				
Reference	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *BaseObjectType* from Part 3 – Address Space Model.					
HasProperty	Variable	Pressure	Double	PropertyType	Mandatory
HasProperty	Variable	Temperature	Float	PropertyType	Mandatory
HasProperty	Variable	Flow	Double	PropertyType	Optional

The *BrowseName* is a non-localised name for an *ObjectType*.

IsAbstract is a flag indicating whether instances of the *ObjectType* can be created. If *IsAbstract* is FALSE then instances of this *ObjectType* may be created. If *IsAbstract* is TRUE then instances of the *ObjectType* cannot be created, the *ObjectType* must be subtyped.

The bottom of the table lists the child *Nodes* for the type. The *Reference* column is the type of *Reference* between the *Object* instance and the child *Node*. The *NodeClass* is the class of *Node*.

The *BrowseName* is the non-localised name for the child. The *DataType* is the structure of the Value accessible via the *Node* (only used for *Variable NodeClass Nodes*) and the *TypeDefinition* is the *ObjectType* or *VariableType* for the child.

The *ModellingRule* indicates whether a child is *Mandatory* or *Optional*. It can also indicate cardinality. Note that the *BrowseName* is not defined if the cardinality is greater than 1. Figure 6 visually depicts the *ObjectType* defined in Table 3 along with two instances of the *ObjectType*. The first instance includes the *Optional Property* while the second does not.

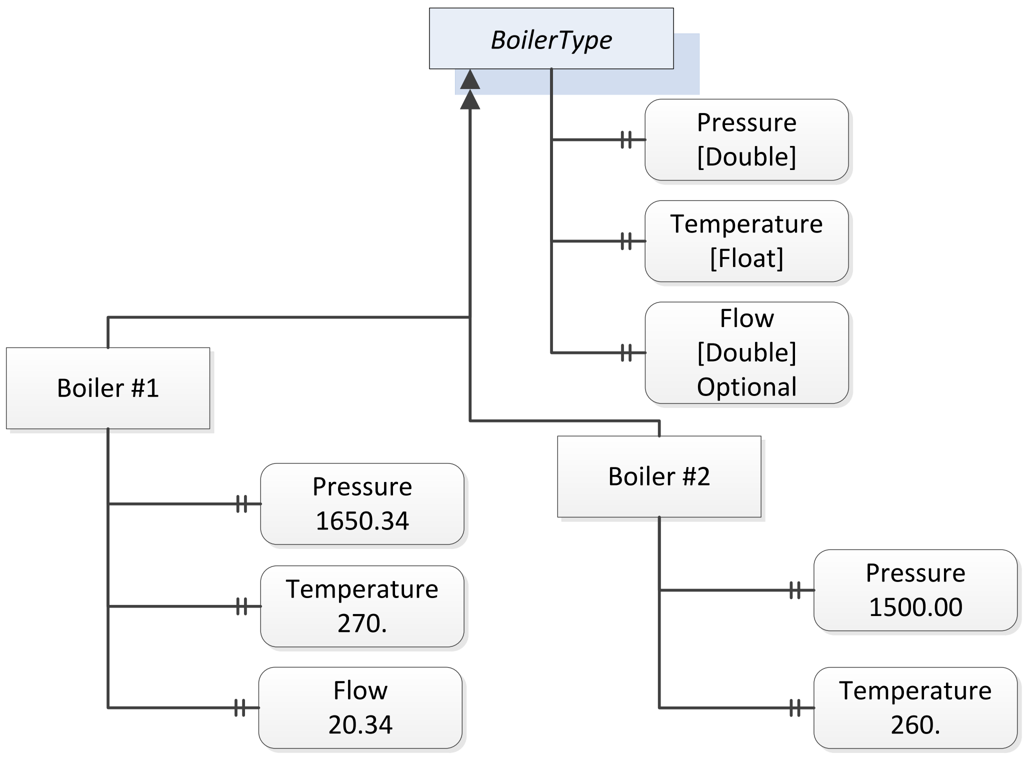

Figure 6 - A Visual Representation of the Sample ObjectType

4 Architectures

4.1 Overview

The following section describes the two architectures that are defined by this specification. The *Object* models defined in other sections of this specification are affected by these architectures (see Figure 7.)

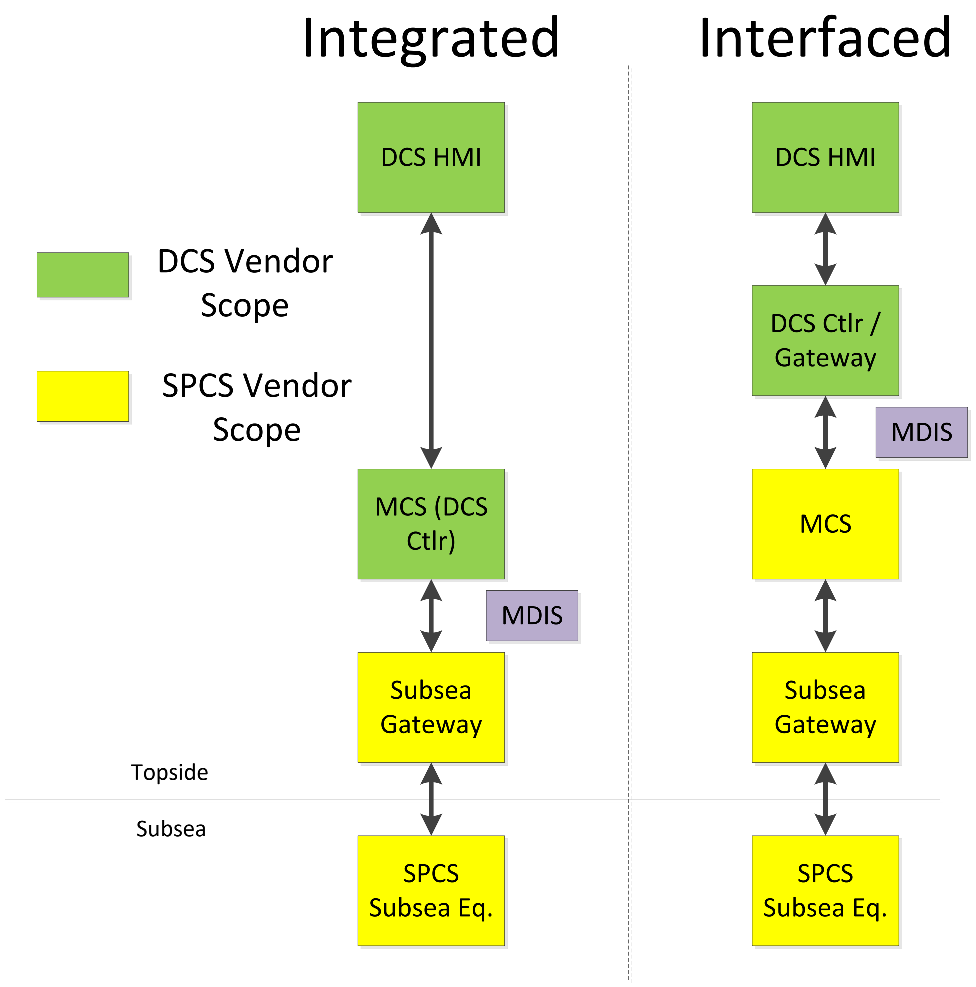

Figure 7 - Architecture Overview

This narrative and associated architecture drawing are intended to identify and represent this interface in the majority of typical system implementations. It is not intended to mandate the detailed architecture of a DCS vendor or SPCS vendor's control system, nor is it intended to suggest or exclude a particular contracting / commercial strategy. This simplified version of the MDIS interface was used to facilitate development of the data objects and to define the data content between the DCS and SPCS vendor's system.

Two major architectures, "Integrated" or "Interfaced", are typically used througho ut the industry and the choice will typically be decided by the Operating Company. Since the control aspects of the subsea system can be accomplished by both the DCS system or by the subsea system, the actual interface between the two systems may be different. In the Integrated architecture (Case 1), the controls system is an integrated system where all control is performed by the DCS vendor's hardware and the standard needs to support communication of all information between the subsea gateway and the DCS control system. This enables a single HMI (or set of HMIs) to control and monitor platform and subsea operations. In the Interfaced architecture (Case 2), the SPCS vendor provides the controls for the subsea aspects of the system and the DCS system is used for monitoring and set point control purposes of the subsea system, along with topside controls. The MDIS *InformationModel* is able to adapt to both of these architectures.

4.2 DCS Implemented Functions

4.2.1 Main Process Responsibility

The DCS is the primary user interface to the overall facility process including the subsea system. Process data management is handled by the DCS as well as all process alarming, alarm management and event / data archiving. Although an MCS may have an alarm or event queue, the primary facility alarm management occurs at the DCS level. Access to the various subsea control functions are managed by the DCS user access level rather than in the subsea system. The DCS also serves as the master for time synchronisation (for addition details see section 10).

4.2.2 Control and Monitoring of Subsea Equipment

Normal control and monitoring of the subsea production system is conducted at the DCS HMI. There may be a separate maintenance or configuration workstation used by the SCV, but it is not within the scope of the MDIS interface.

4.2.3 Subscriptions

OPC UA supports *Subscription* and polling (*Read*) manners of obtaining data. The *Subscription* based manner of obtaining data should be used by default. *Subscription*, which is exception reporting, typically provides improved performance over the polling interface.

4.3 DCS or SCV Implemented Functions

4.3.1 Introduction

The functional elements of the system either reside in the DCS or SPCS depending on a particular vendor's solution or customer's requirements. The "Operating Company" should specify where each of these optional functions should reside. See Figure 8 for an illustration.

4.3.2 Data Arbitration

Data Arbitration is the system function that manages the reception and transmission of dual / redundant SPCS data.

If the Subsea system performs this function, only a single process value or operator command is typically passed between the SPCS and DCS system. If the DCS performs this function, both the A and B data values would typically be passed across the interface.

There are multiple types of data that could require arbitration. Instruments can be redundant, SEMs can be redundant and it is possible that the different types of data maybe arbitrated in different locations. I.e. in some projects, sensor data may be arbitrated by the DCS while the SEM may be arbitrated by the SPCS. Data Arbitration choices can also affect redundancy.

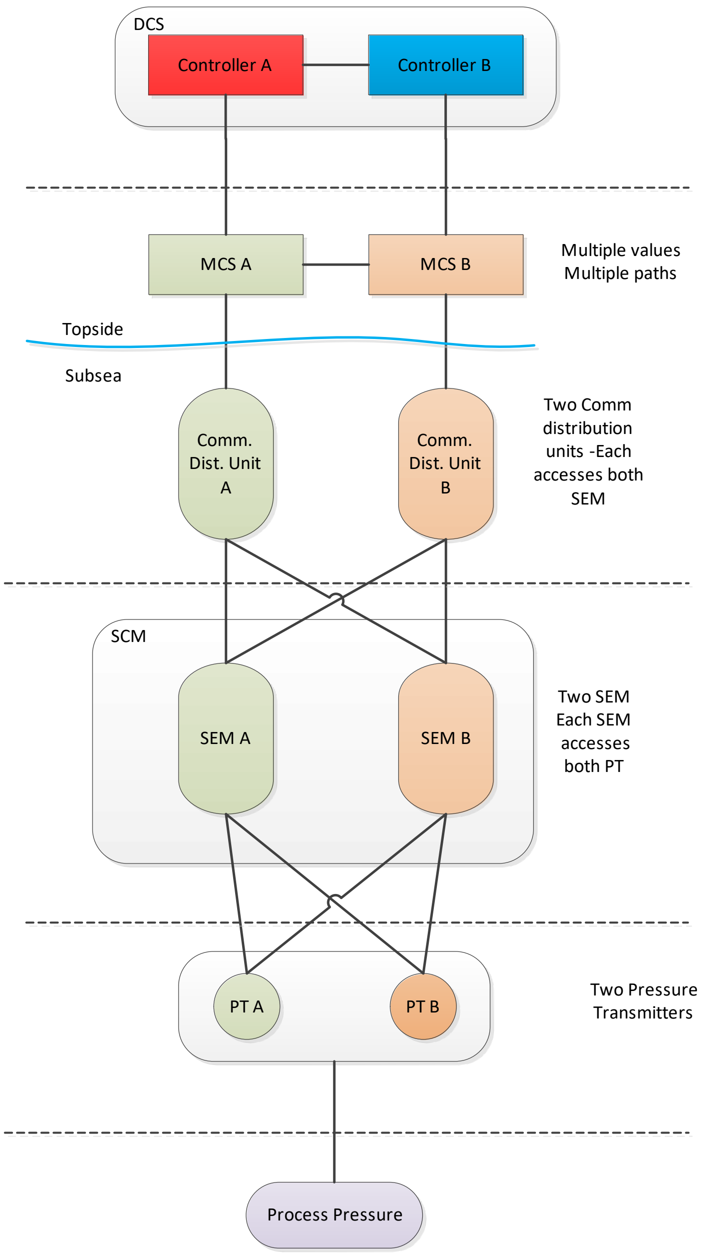

Figure 8 – Data Arbitration Example

4.3.3 SEM Control Selection

Certain subsea instruments may only be powered by one SEM at a time, selectable by the operator. Also, a SEM may have various modes, such as ROV mode or maintenance mode, which can be selected.

4.3.4 Interlocks

4.3.4.1 Introduction

An Interlock is a control permissive that exists to prevent or warn an operator against potentially undesired operator commands being issued to the subsea system. Depending on an operator's access level, he / she may be able to override the interlock in order to perform the desired operation. Interlocks can be categorised into two types: process interlocks and product / system interlocks, though not all customers or SCV's make this distinction.

4.3.4.2 Process Interlocks

Process interlocks are interlocks which are specific to a particular project dependent on field layout, tree functionality, etc. These are often defined by the customer's process requirements or by regulatory agencies; e.g., prevention of opening the tree crossover valve if the production master valve and annulus master valve are open

4.3.4.3 Product or System Interlocks

Interlocks defined by the SCV for the protection of the subsea system; for example, low hydraulic pressure inhibiting opening (pressurising) of a tree valve. These interlocks are typically not able to be overridden by an operator.

4.3.5 Shutdown Sequences

These are defined subsea valve operation sequences that take the subsea system to a safe state. They are initiated either by subsea process conditions, operator intervention or emergency conditions triggered from external interfaces such as the facility Safety Instrumented System (SIS).

4.3.6 Automated Control Sequences

These are multi-step control sequences triggered by the issuance of a single operator command, such as smart well (interval control valve) controls, hydrate prevention or preparation of a tree for start-up.

4.3.7 Determining Valve Statuses

This refers to determination of the status of a subsea valve by evaluating some or all of the following: hydraulic output function line pressure, hydraulic flow and last command received.

4.3.8 Determining / Updating Choke Calculated Position

This refers to the calculation of the assumed choke position based upon the number of step commands issued to the subsea choke. It may be maintained in percentage open or step position and is compared to the position transducer on the choke for calibration.

4.3.9 HPU Interface

The HPU interface may include HPU control capability, data monitoring and configuration such as pump control setpoint changes.

4.3.10 EPU Interface

The interface to the EPU may include monitoring of the power supply to the subsea equipment including input voltage / current, umbilical voltage(s) / current(s), line insulation monitoring data and power alarm statuses (over-voltage and over-current).

4.3.11 Valve Profile / Signature Validation

A valve profile, or signature, is a representation of the performance of a subsea valve in terms of its hydraulic fluid pressure and flow characteristics as measured at the subsea control module. Valve Profile / Signature Validation is a software function that compares a current valve profile/signature to a baseline or template signature recorded previously, typically at subsea system commissioning. Not all systems have this functionality.

4.3.12 Topsides Chemical Injection System Interface

The chemical injection interface may include control and monitoring capability. Typically, the interface includes verification to the subsea system of chemical delivery (flow rate and / or pressure) from the topsides chemical injection system.

4.4 Subsea Controls Vendor-Implemented Functions

4.4.1 Introduction

These functions are assumed to be always implemented in the SPCS vendor's equipment. In the case that the MCS is provided by a third-party supplier, the references below to the DCS may also pertain to the MCS.

4.4.2 Managing Subsea Communications

The SCV's system will manage data traffic to and from the subsea system and issue device control commands. The protocol is typically proprietary for a particular SCV and the medium and redundancy requirements are dependent upon customer requirements. The interface from the subsea gateway to the subsea system is not within the scope of the MDIS interface.

4.4.3 Operation of Subsea Devices

Ultimate operation or actuation of a subsea device is executed by the SCV's system, whether requested locally, such as from an SCV engineering workstation, or remotely from the DCS.

4.4.4 Handing off Process Sensor Data to DCS

The SCV's system will provide current process data (e.g., pressures, temperatures, flow rates) and statuses (e.g., valve positions) to the DCS.

4.4.5 Configuration of Operational Parameters

This includes settings for low-level subsea system functionality, such as solenoid pulse timers, pressure check settings for evaluating valve position or unintended movement, timer setpoints for determining valve failure, etc.

4.4.6 Handing off Valve Profiles / Signatures

Valve *Profiles* are made available for transmission from the SCV system. The output format may vary among vendors and the data may be transmitted according to customer requirements.

4.4.7 Calculation of Engineering Values

The SCV system typically calculates process engineering values if raw data is received from subsea devices, though there may be exceptions where raw data transmission is required.

4.4.8 Handing off Product Statuses

This refers to any available data in the subsea system not included within the definition of other objects that may be transmitted via a "generic" discrete or analogue object. This includes data that may have been considered "alarms" in legacy subsea systems, but are simply data points that are available to the DCS to manage as alarms, events or to be logged as desired. The SCV may also implement "roll-up" statuses that condense numerous statuses into fewer bits / words in order to optimise data transfer.

4.4.9 Handing Off Diagnostic Information

Diagnostic information in regard to the health of the subsea system is managed in the SCV's system. This data would typically not be transmitted to the DCS except for summary product status data as defined above. It would be transmitted via a "generic" discrete or analogue object as desired.

4.4.10 EPU Interface

The interface to the EPU may include monitoring of the power supply to the subsea equipment including input voltage / current, umbilical voltage(s) / current(s), line insulation monitoring data and power alarm statuses (over-voltage and over-current).

4.4.11 Subsea Control Paths / Network Routing

The SCV defines the subsea communications system architecture. Communications link control and monitoring is also performed by the SCV. Variable scan configurations (e.g. fast scan, normal scan, slow scan) may be implemented and configured by the SCV as required.

5 MDIS ObjectTypes

5.1 Overview

The following sections define the basic OPC UA *Objects* defined by MDIS. This includes *Method* definition as needed. The use cases / object interactions for each *Object* are defined in a separate section.

5.1.1 MDISBaseObjectType

The *MDISBaseObjectType* is a base object that all other MDIS objects are constructed from. It is an abstract *ObjectType* and instances of it shall not exist. This *Object* will be used to create subtypes.

5.1.2 MDISDiscreteInstrumentObjectType

The *MDISDiscreteInstrumentObjectType* is a base type and can be subtyped or instances of it can be directly created. The *Object* can be used with multi-state type of data (stopped, moving, faulted). It could also be used for integer values from instruments. For a limit switch or on / off switch the *MDISDigitalInstrumentObjectType* should be used.

5.1.3 MDISDiscreteOutObjectType

The *MDISDiscreteOutObjectType* is a subtype of *MDISDiscreteInstrumentObjectType* and can be subtyped or instance of it can be directly created. The *Object* can be used for Tristate or Multistate switches.

5.1.4 MDISDigitalInstrumentObjectType

The *MDISDigitalInstrumentObjectType* is a base type and can be subtyped or instance of it can be directly created. The *Object* can be used to represent on / off type of functions.

5.1.5 MDISDigitalOutObjectType

The *MDISDigitalOutObjectType* is a subtype of *MDISDigitalInstrumentObjectType* and can be subtyped or instance of it can be directly created. The *Object* can be used for switching on / off types.

5.1.6 MDISInstrumentObjectType

The *MDISInstrumentObjectType* is a base type and can be subtyped or instances of it can be directly created. The *Object* can be used for various types of analogues, e.g. pressure, temperatures, tank levels etc.

5.1.7 MDISInstrumentOutObjectType

The *MDISInstrumentOutObjectType* is a subtype of *MDISInstrumentObjectType* and can be subtyped or instance of it can be directly created. The *Object* can be used for writing floating point values.

5.1.8 MDISChokeObjectType

The *MDISChokeObjectType* object is a base type and can be subtyped or an instance of it can be directly created. A choke is a device that restricts the flow of a fluid (gases, liquids, fluidised solids, or slurries).

5.1.9 MDISValveObjectType

The *MDISValveObjectType* object is a base type and can be subtyped or an instance of it can be directly created. A valve is a device that directs or controls the flow of a fluid (gases, liquids, fluidised solids, or slurries). The *MDISValveObjectType* represents a two state valve type.

5.1.10 MDISAggregateObjectType

An abstract type that all aggregate *ObjectTypes* shall be derived from. This *ObjectType* allows *Clients* to easily identify aggregate *Objects*. For more information about aggregation see 9.5

5.1.11 MDISTimeSyncObjectType

The *MDISTimeSyncObject* (see 5.9.3) is a base *ObjectType*. An instance of this *ObjectType* shall be exposed as part of the *MDISInformationObjectType,*if the *MDISTimeSyncObjectType* is supported.

5.1.12 MDISInformationObjectType

The *MDISInformationObjectType* (see 5.10) is a base *ObjectType*. An instance of this *ObjectType* shall be exposed under the *Objects* folder. It provides information about the MDIS Information model that is supported by the Server. It can also expose additional information related to MDIS.

5.2 MDISBaseObjectType

5.2.1 Overview

The following section details the MDIS generic properties for the *MDISBaseObjectType*. Implementations shall ensure adherence to *Mandatory* [M] aspects in order to comply with the MDIS interface standardisation. *Optional* [O] may or may not be implemented within a project. Figure 9 provides an overview of the *MDISBaseObjectType* as defined by MDIS. This *Object* is intended to be the base object for all other MDIS *ObjectTypes* (see Figure 10 for an overview of inherited types)

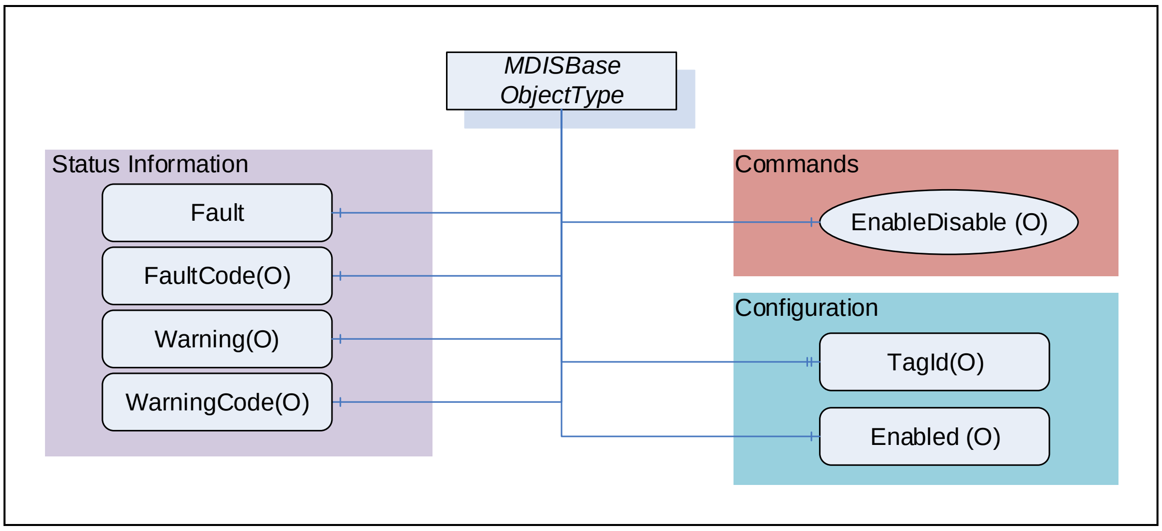

Figure 9 - MDISBaseObjectType

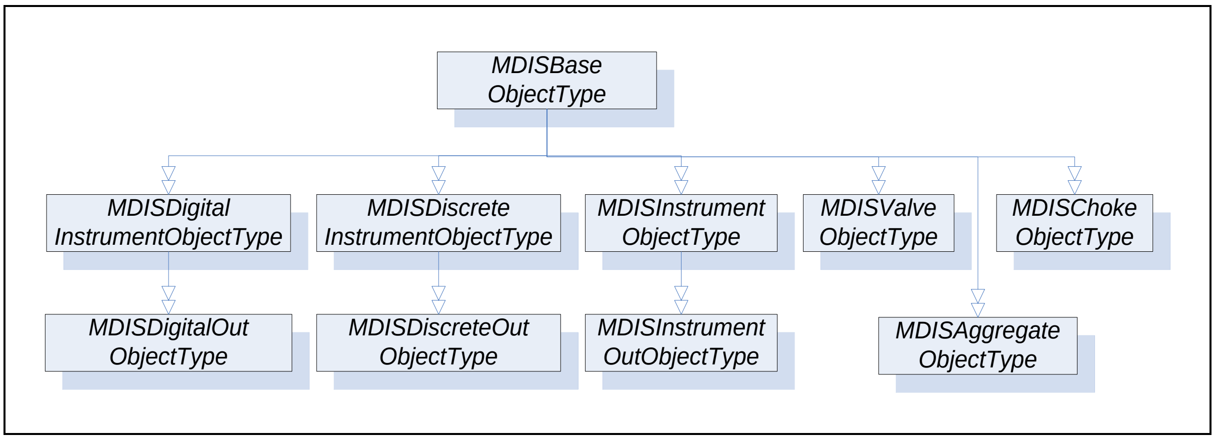
MDISBase ObjectType
MDISDigital InstrumentObjectType
MDISDiscrete InstrumentObjectType
MDISInstrument ObjectType
MDISValve ObjectType
MDISChoke ObjectType
MDISDigitalOut ObjectType
MDISDiscreteOut ObjectType
MDISInstrument OutObjectType
MDISAggregate ObjectType

Figure 10 - Base Object Hierarchy

5.2.2 MDISBaseObjectType Definition

The Table 4 defines the structure of an *MDISBaseObjectType*.

Table 4 - MDISBaseObjectType

Attribute	**Value**					
BrowseName	MDISBaseObjectType					
IsAbstract	True					
References	**Node Class**	**BrowseName**	**Data Type**	**TypeDefinition**	**Modelling Rule**	**RW**
Subtype of the BaseObjectType defined in OPC UA Part 5 – Information Model						
HasComponent	Variable	Fault	Boolean	BaseDataVariableType	Mandatory	R
HasComponent	Variable	Warning	Boolean	BaseDataVariableType	Optional	R
HasComponent	Variable	Enabled	Boolean	BaseDataVariableType	Optional	R
HasProperty	Variable	TagId	String	PropertyType	Optional	R
HasComponent	Method	EnableDisable	See 5.2.3		Optional	
HasComponent	Variable	FaultCode	UInt32	BaseDataVariableType	Optional	R
HasComponent	Variable	WarningCode	UInt32	BaseDataVariableType	Optional	R
HasSubtype		MDISDigitalInstrumentObjectType				
HasSubtype		MDISDiscreteInstrumentObjectType				
HasSubtype		MDISChokeObjectType				
HasSubtype		MDISInstrumentObjectType				
HasSubtype		MDISValveObjectType				
HasSubtype		MDISAggregateObjectType				

The RW column indicates if a *Node* of *Variable NodeClass* is readable, writeable or both readable and writeable. Other *NodeClasses* (*Object*, *Method*) do not support reading or writing and do not fill in this column.

By definition a *Profile* can require that an *Optional* item be provided, it cannot change the behaviour of an *Object* from what is described in this specification, which includes support for any *Mandatory* items. *Profiles* are described in section 12.

Fault – The status of the object, true if any fault exists.

Warning – The status of the object, true if any warnings exist. A warning does not require immediate operator action.

Enabled – This *Variable* is set as enabled (true) by default. When disabled the *Object* will not report any dynamic information other than a bad status code (*Bad_InvalidState*). It will still report configuration related information. For the MDISBaseObjectType the default is that only the Enabled flag, TagId and Enable method report values or perform functions. Subtypes of this *ObjectType* may describe additional requirements for disabled *Objects*.

TagId – The *TagId* is a unique equipment identifier. This is additional information that can be used to help identify the *Variable* associated with the instance of this type. This field is intended to be used to store the tag id from the P&ID

EnableDisable – This method allows a *Client* to disable or enable the *Object*.

FaultCode – An unsigned integer that describes a fault code(s), zero indicates no fault. The SPCS vendor will provide a definition of what the number means. It might be a bit field or a fault code.

WarningCode – An unsigned integer that describes a warning code(s), zero indicates no warning. The SPCS vendor will provide a definition of what the number means. It might be a bit field or an error code. If a WarningCode is provided then the Warning flag shall also be provided.

5.2.3 EnableDisable Method

EnableDisable is used to disable or enable an *Object*. The enable / disable operation applies to the *Object* in the UA *Server*. The call completes when the enable / disable operation is complete. The *Server* may or may not pass the enable / disable down to lower levels. This is *Server* specific behaviour.

Method Declaration

```
EnableDisable (
      [in] Enable    Boolean
);
```

Table 5 - EnableDisable Method parameters

Argument	Description
Enable	Boolean indicator of whether the *Object* is to be disabled or enabled. A true indicates that the *Object* is enabled.

Method result codes are defined as part of the *Call Service* (see OPC UA Services Part 4 – Services specification). They are described in Table 93 for ease of reference.

Comments

The *EnableDisable Method* will disable or enable this *Object*. Once the state of an *Object* is changed by this *Method* (i.e. disabled) the state will be maintained until this *Method* is called again to change the state (i.e. enable). The *Method* will report if any error occurs while disabling or enabling the *Object*. Table 6 specifies the *AddressSpace* representation for the *EnableDisable Method*.

Table 6 - EnableDisableMethod AddressSpace Definition

Attribute	**Value**				
BrowseName	Disable				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory

5.3 MDISDiscreteInstrumentObjectType

5.3.1 Introduction

The following section details the generic *MDISDiscreteInstrumentObjectType* structure and defines the properties associated with it. Additional sections define a subtype *MDISDiscreteOutObjectType* that allows updates to the discrete value. This is in general a vendor and operator independent description, but all users of the *MDISDiscreteInstrumentObjectType* or *MDISDiscreteOutObjectType* can add vendor specific data. The vendor specific data should be defined as part of a subtype of the *MDISDiscreteInstrumentObjectType* or *MDISDiscreteOutObjectType* defined in this document. It is assumed that the subsea system is the *Server* and host of the instance of the *MDISDiscreteInstrumentObjectType* or *MDISDiscreteOutObjectType*. The DCS based system is the *Client* in the system. It is assumed that all interactions with the instance of the *MDISDiscreteInstrumentObjectType* are initiated by the *Client* and are directed to the *Server*.

5.3.2 Overview

The following section details the MDIS generic properties for the *MDISDiscreteInstrumentObjectType*. Implementation shall ensure adherence to *Mandatory* [M] aspects in order to comply with the MDIS interface standardisation. *Optional* [O] may or may not be implemented within a project. Figure 11 provides an overview of the *MDISDiscreteInstrumentObjectType* as defined by MDIS, including some nested types. This figure includes all items that are inherited from the *MDISBaseObjectType*.

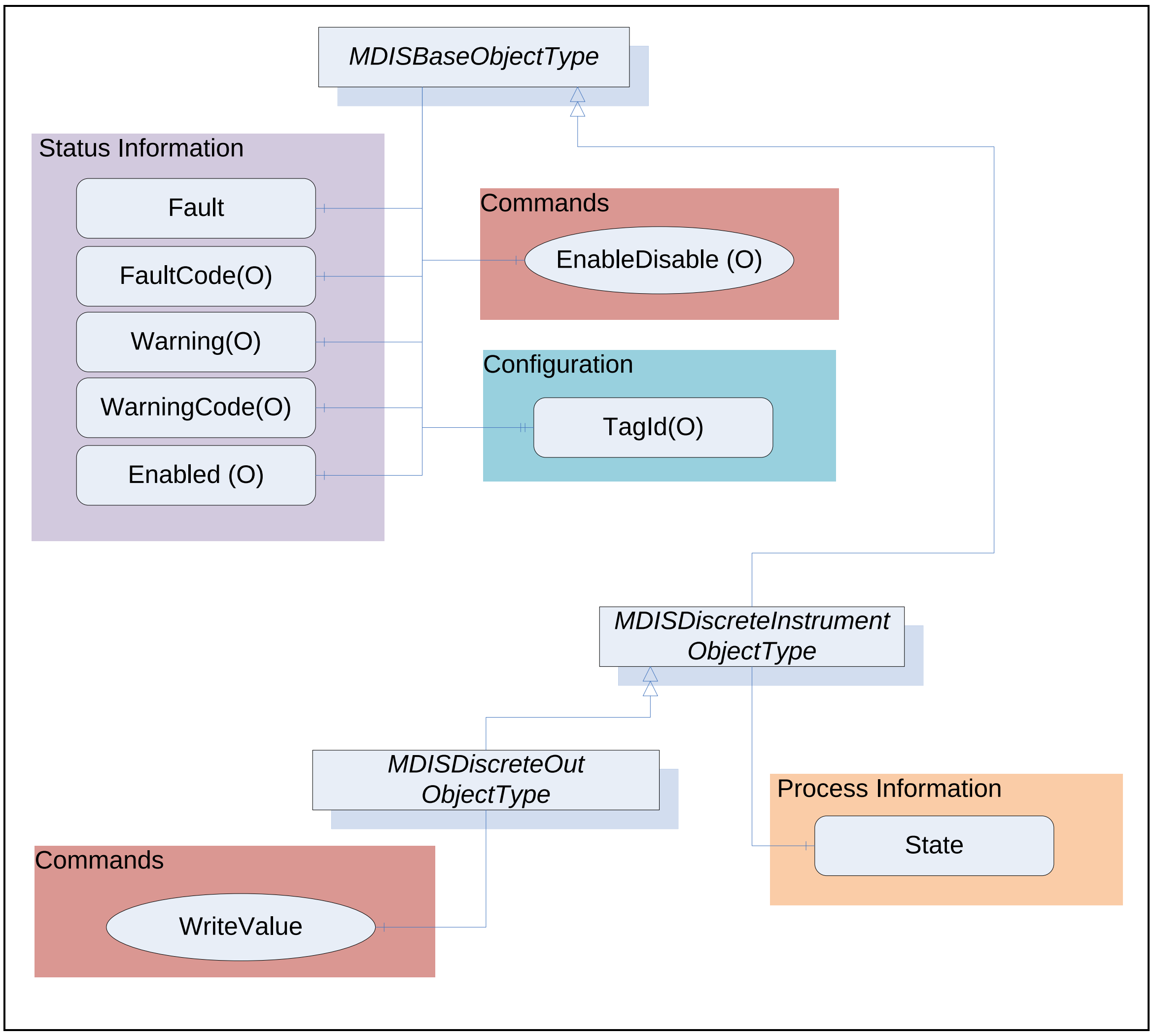

Figure 11 - MDISDiscreteInstrumentObjectType & MDISDiscreteOutObjectType

5.3.3 MDISDiscreteInstrumentObjectType Definition

Table 7 defines the structure of an *MDISDiscreteInstrumentObjectType*. Any vendor specified properties that have been implemented within a project should be documented within a similar format and supplied to the DCS vendor. The addition of vendor specific properties will result in a subtype of the *MDISDiscreteInstrumentObjectType*.

Table 7 - MDISDiscreteInstrumentObjectType

Attribute	**Value**					
BrowseName	MDISDiscreteInstrumentObjectType					
IsAbstract	False					
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**	**RW**
Subtype of the MDISBaseObjectType (see section 5.1.1)						
HasComponent	Variable	State	UInt32	BaseDataVariableType	Mandatory	R
HasSubtype	MDISDiscreteOutObjectType					

State – The state of the instance of *MDISDiscreteInstrumentObjectType*. This state is represented as a UInt32.

5.3.4 MDISDiscreteOutObjectType Definition

Table 8 defines the structure of an *MDISDiscreteOutObjectType*. Any vendor specified properties that have been implemented within a project should be documented within a similar format and supplied to the DCS vendor. The addition of vendor specific properties will result in a subtype of the *MDISDiscreteOutObjectType*.

Table 8 - MDISDiscreteOutObjectType

Attribute	Value					
BrowseName	MDISDiscreteOutObjectType					
IsAbstract	False					
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**	**RW**
Subtype of the MDISDiscreteInstrumentObjectType (see section 5.1.2)						
HasComponent	Method	WriteValue	See 5.3.5		Mandatory	

WriteValue – This *Method* allows a *Client* to change the value of *State* on an instance of *MDISDiscreteOutObjectType*. The *Method* will return any errors that occurred on setting the value. The *Client* shall verify that the *State* actually changed to confirm the update.

5.3.5 WriteValue Method

WriteValue Method (defined in Table 9) is used to change the value of the *State Variable* in an instance of the *MDISDiscreteOutObjectType*. The *WriteValue* operation applies to the object in the subsea system. Some systems will be able to report any errors immediately others will only be able to report that the operation was not refused. *Clients* are expected to monitor the *State* and ensure that the operation completed. If an error occurs after the *Method* has returned, a *Fault* flag shall be set and an appropriate *FaultCode* will be returned. The *Fault* (and *FaultCode*) will reset on the next successful *WriteValue Method* invocation.

Method Declaration

WriteValue (

```
   [in] State    UInt32
);
```

Table 9 - WriteValue Method parameters

Argument	Description
State	UInt32 value *Variable*, that indicates the target state of the *Variable*

Method result codes are defined as part of the *Call Service* (see OPC UA Services Part 4 – Services specification). They are described in Table 93 for ease of reference.

Comments

The *WriteValue Method* will change the value of the *State Variable*. The *Method* will report if any error occurs while writing the state of the *Object*. Table 10 specifies the *AddressSpace* representation for the *WriteValue Method*.

Table 10 - WriteValue Method AddressSpace Definition

Attribute	Value				
BrowseName	State				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory

5.4 MDISDigitalInstrumentObjectType

5.4.1 Introduction

The following section describes the generic *MDISDigitalInstrumentObjectType* structure and defines the properties associated with it. Additional sections define a subtype *MDISDigitalOutObjectType* that allows updates to the digital value. This is in general a vendor and operator independent description, but all users of the *MDISDigitalInstrumentObjectType* or *MDISDigitalOutObjectType* can add vendor specific data. The vendor specific data should be defined as part of a subtype of the *MDISDigitalInstrumentObjectType* or *MDISDigitalOutObjectType* defined in this document. It is assumed that the subsea system is the *Server* and host of the instance of *MDISDigitalInstrumentObjectType* or *MDISDigitalOutObjectType*. The DCS based system is the *Client* in the system. It is assumed that all interactions with the instance of the *MDISDigitalInstrumentObjectType* are initiated by the *Client* and are directed to the *Server*.

5.4.2 Overview

The following section details the MDIS generic properties for the *MDISDigitalInstrumentObjectType*; implementation shall ensure adherence to *Mandatory* [M] aspects in order to comply with the MDIS interface standardisation. *Optional* [O] may or may not be implemented within a project. Figure 12 provides an overview of the *MDISDigitalInstrumentObjectType* as defined by MDIS, including some nested types. This figure includes all items that are inherited from the *MDISBaseObjectType*.

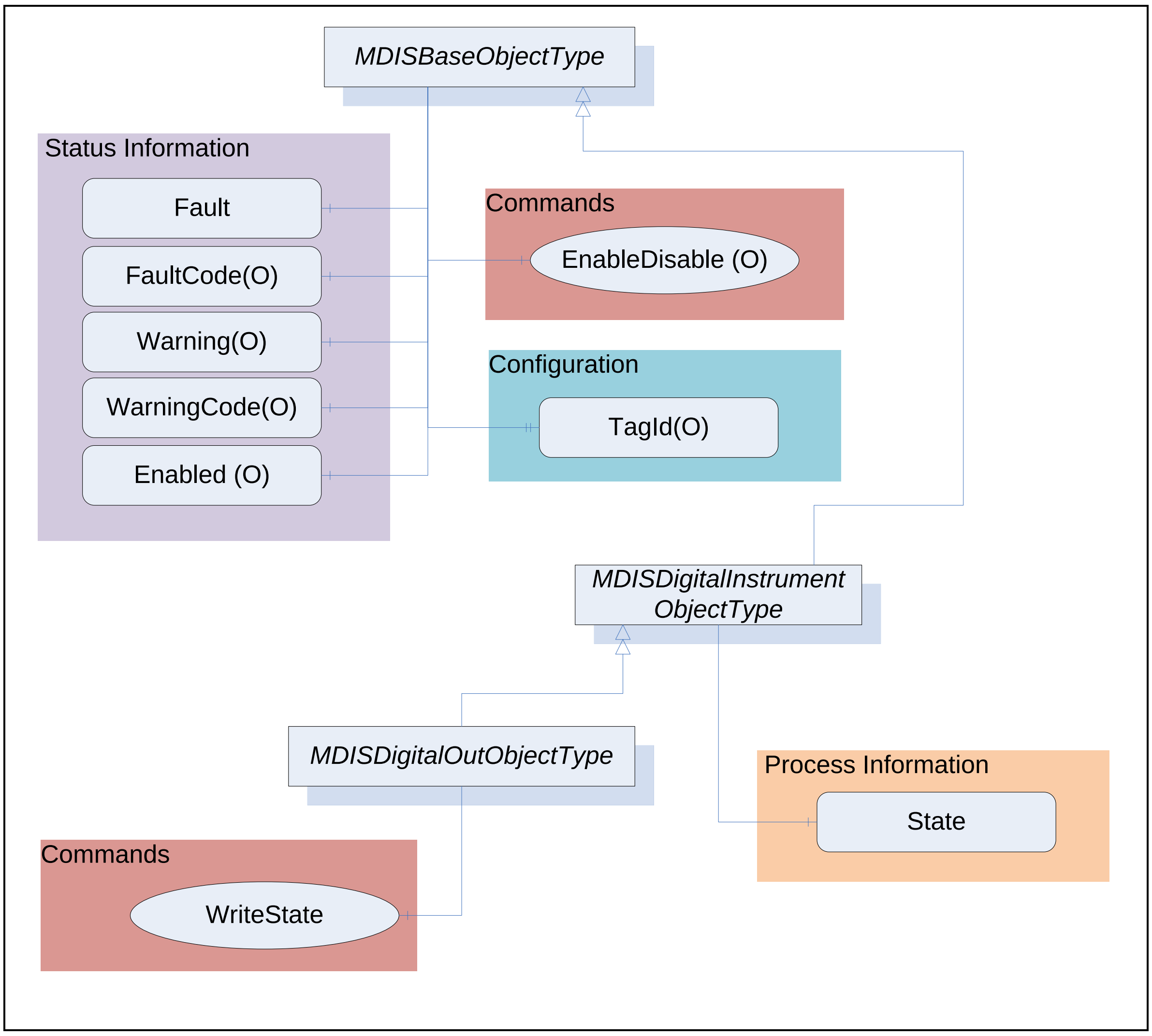

Figure 12 - MDISDigitalInstrumentObjectType & MDISDigitalOutObjectType

5.4.3 MDISDigitalInstrumentObjectType Definition

Table 11 defines the structure of an *MDISDigitalInstrumentObjectType*. Any vendor specified properties that have been implemented within a project should be documented within a similar format and supplied to the DCS vendor. The addition of vendor specific properties will result in a subtype of the *MDISDigitalInstrumentObjectType*.

Table 11 - MDISDigitalInstrumentObjectType

Attribute	**Value**					
BrowseName	MDISDigitalInstrumentObjectType					
IsAbstract	False					
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**	**RW**
Subtype of the MDISBaseObjectType (see section 5.1.1)						
HasComponent	Variable	State	Boolean	BaseDataVariableType	Mandatory	R
HasSubtype	MDISDigitalOutObjectType					

State – The state of the instance of *MDISDigitalInstrumentObjectType*. This state is represented as a Boolean, where true indicates on and false indicates off.

5.4.4 MDISDigitalOutObjectType

Table 12 defines the structure of an *MDISDigitalOutObjectType*. Any vendor specified properties that have been implemented within a project should be documented within a similar format and supplied to the DCS Vendor. The addition of vendor specific properties will result in a subtype of the *MDISDigitalOutObjectType*.

Table 12 - MDISDigitalOutObjectType

Attribute	**Value**					
BrowseName	MDISDigitalOutObjectType					
IsAbstract	False					
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**	**RW**
Subtype of the MDISDigitalInstrumentObjectType						
HasComponent	Method	WriteState	See 5.4.5		Mandatory	

WriteState – This *Method* allows a *Client* to change the value of *State* on an instance of *MDISDigitalOutObjectType*. The *Method* will return any errors that occurred on setting the value. The *Client* shall verify that the *State* actually changed to confirm the update.

5.4.5 WriteState Method

WriteState Method (defined in Table 13) is used to change the state of the *State Variable* in an instance of the *MDISDigitalOutObjectType*. The *WriteState* operation applies to the object in the subsea system. Some systems will be able to report any errors immediately others will only be able to report that the operation was not refused. *Clients* are expected to monitor the *State* and ensure that the operation completed. If an error occurs after the *Method* has returned, a *Fault* flag shall be set and an appropriate *FaultCode* will be returned. The *Fault* (and *FaultCode*) will reset on the next successful *WriteState Method* invocation.

Method Declaration

WriteState (

```
   [in] State    Boolean
);
```

Table 13 – WriteState Method parameters

Argument	**Description**
State	Boolean indicator of the target state of the variable

Method result codes are defined as part of the *Call Service* (see OPC UA Services Part 4 – Services specification). They are described in Table 93 for ease of reference.

Comments

The *WriteState Method* will change the state of the *State Variable*. The *Method* will report if any error occurs while writing the state of the *Object*. Table 14 specifies the *AddressSpace* representation for the *WriteState Method*.

Table 14 - WriteState Method AddressSpace Definition

Attribute	**Value**				
BrowseName	State				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory

5.5 MDISInstrumentObjectType

5.5.1 Introduction

The following section details the generic *MDISInstrumentObjectType* structure and defines the properties associated with it. Additional sections define a subtype *MDISInstrumentOutObjectType* that allows updates to the instrument value. This is in general a vendor and operator independent description, but all users of the *MDISInstrumentObjectType* or *MDISInstrumentOutObjectType* can add vendor specific data. The vendor specific data should be defined as part of a subtype of the *MDISInstrumentObjectType* defined in this document. It is assumed that the subsea system is the *Server* and host of the instance of the *MDISInstrumentObjectType* or *MDISInstrumentOutObjectType*. The DCS based system is the *Client* in the system. It is assumed that all interactions with the *MDISInstrumentObjectType* are initiated by the *Client* and are directed to the *Server*.

5.5.2 Overview

The following section details the MDIS generic properties for the *MDISInstrumentObjectType*. Implementation shall ensure adherence to *Mandatory* [M] aspects in order to comply with the MDIS interface standardisation. *Optional* [O] may or may not be implemented within a project. Figure 13 provides an overview of the *MDISInstrumentObjectType* as defined by MDIS, including some nested types. Figure 13 includes all of the items that are inherited from the *MDISBaseObjectType*.

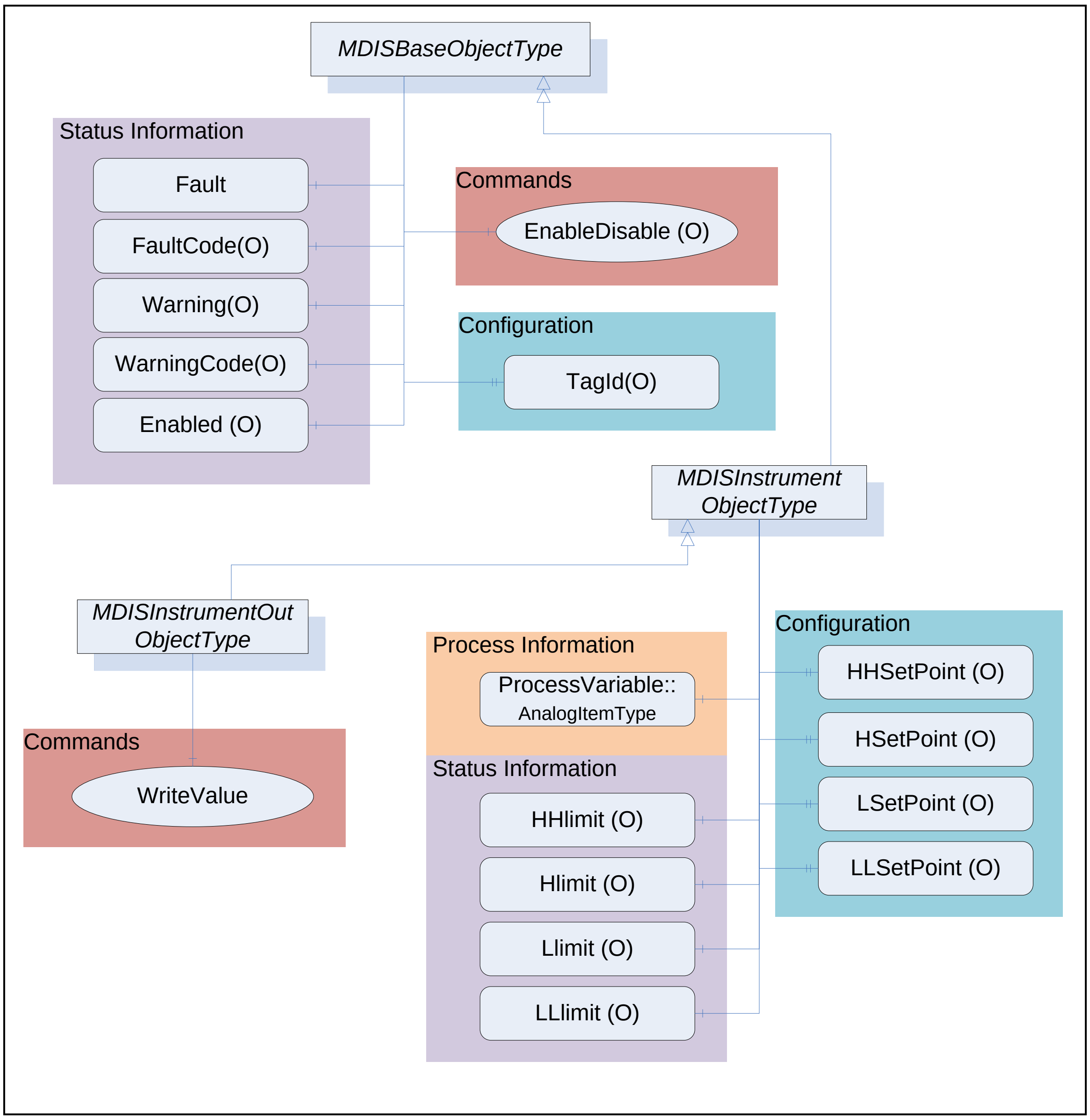

Figure 13 - MDISInstrumentObjectType & MDISInstrumentOutObjectType

5.5.3 MDISInstrumentObjectType Definition

Table 15 defines the structure of an *MDISInstrumentObjectType*. Any vendor specified properties that have been implemented within a project should be documented within a similar format and supplied to the DCS vendor. The addition of vendor specific properties will result in a subtype of the *MDISInstrumentObjectType*. If a MDISInstrumentObjectType instance is disabled, the MDISBaseObjectType defaults are followed and only the HHSetPoint,HSetpoint,LSetpoint and LLSetpoint object values will be available

Table 15 - MDISInstrumentObjectType

Attribute	**Value**					
BrowseName	MDISInstrumentObjectType					
IsAbstract	False					
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**	**RW**
Subtype of the MDISBaseObjectType (defined in 5.1.1)						
HasComponent	Variable	ProcessVariable	Float	AnalogItemType (see Part 8 – Data Access)	Mandatory	R
HasComponent	Variable	HHlimit	Boolean	BaseDataVariableType	Optional	R
HasComponent	Variable	Hlimit	Boolean	BaseDataVariableType	Optional	R
HasComponent	Variable	Llimit	Boolean	BaseDataVariableType	Optional	R
HasComponent	Variable	LLlimit	Boolean	BaseDataVariableType	Optional	R
HasProperty	Variable	HHSetPoint	Float	PropertyType	Optional	RW
HasProperty	Variable	HSetPoint	Float	PropertyType	Optional	RW
HasProperty	Variable	LSetPoint	Float	PropertyType	Optional	RW
HasProperty	Variable	LLSetPoint	Float	PropertyType	Optional	RW

ProcessVariable – a *Variable* in engineering units that represents the value of the instance of an *MDISInstrumentObjectType*. It includes properties that represent the engineering units; the engineering units range and optionally the instrument range, see Table 16 for an illustration, for actual definition see *AnalogItemType* in OPC UA Part 8. Both the engineering units and engineering units' range are required.

Table 16 – *AnalogItemType* definition

Attribute	**Value**				
BrowseName	AnalogItemType				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InstrumentRange	Range	PropertyType	Optional
HasProperty	Variable	EURange	Range	PropertyType	Mandatory
HasProperty	Variable	EngineeringUnits	EUInformation	PropertyType	Mandatory

The EUInformation DataType is defined in Part 8 – Data Access

HHlimit – The instrument HH state is active

Hlimit – The instrument H state is active

Llimit – The instrument L state is active

LLlimit – The instrument LL state is active

HHSetPoint – Configuration of *HHSetPoint* which will set *HHlimit* be TRUE when the *ProcessVariable* value is greater than "set point value". If this limit *Variable* exists on an object, but has not been configured, the *HHSetPoint* shall have a status code of *Bad_ConfigurationError* and *Clients* shall ignore the value. When the HHSetPoint has a Status of Bad_configurationError, if the *HHlimit* exists, it shall have a status code of *Bad_ConfigurationError* and the value is ignored.

HSetPoint – Configuration of *HSetPoint* which will set *Hlimit* be TRUE when the *ProcessVariable* value is greater than "set point value". If this limit *Variable* exists on an object, but has not been configured, the *HSetPoint* shall have a status code of *Bad_ConfigurationError* and *Clients* shall ignore the value. When the HSetPoint is ignored, if the *Hlimit* exists, it shall have a status code of *Bad_ConfigurationError* and the value is ignored.

LSetPoint – Configuration of *LSetPoint* which will set *Llimit* be TRUE when the *ProcessVariable* value is less than "set point value". If this limit *Variable* exists on an object, but has not been configured, the *LSetPoint* shall have a status code of *Bad_ConfigurationError* and *Clients* shall ignore the value. When the LSetPoint is ignored, if the *Llimit* exists, it shall have a status code of *Bad_ConfigurationError* and the value is ignored.

LLSetPoint – Configuration of *LLSetPoint* which will set *LLlimit* be TRUE when the *ProcessVariable* value is less than "set point value". If this limit *Variable* exists on an object, but has not been configured, the *LLSetPoint* shall have a status code of *Bad_ConfigurationError* and *Clients* shall ignore the value. When the LLSetPoint is ignored, if the *LLlimit* exists, it shall have a status code of *Bad_ConfigurationError* and the value is ignored.

5.5.4 MDISInstrumentOutObjectType Definition

Table 17 defines the structure of an *MDISInstrumentOutObjectType*. Any vendor specified properties that have been implemented within a project should be documented within a similar format and supplied to the DCS vendor. The addition of vendor specific properties will result in a subtype of the *MDISInstrumentOutObjectType*.

Table 17 - MDISInstrumentOutObjectType

Attribute	**Value**					
BrowseName	MDISInstrumentOutObjectType					
IsAbstract	False					
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**	**RW**
Subtype of the MDISInstrumentObjectType						
HasComponent	Method	WriteValue	See 5.5.5		Mandatory	

WriteValue – This *Method* allows a *Client* to change the value of *ProcessVariable* on an instance of the *MDISInstrumentOutObjectType*. The *Method* will return any errors that occurred on setting the value. If the Instrument is disabled, an error Bad_InvalidState shall be returned. The *Client* shall verify that the *ProcessVariable* actually changed to confirm the update.

5.5.5 Instrument WriteValue Method

Instrument *WriteValue Method* is used to change the value of the *ProcessVariable* in the *MDISInstrumentOutObjectType*. The Instrument *WriteValue Method* operation applies to the object in the subsea system. Some systems will be able to report any errors immediately others will only be able to report that the operation was not refused. *Clients* are expected to monitor the *ProcessVariable* and ensure that the operation completed successfully. If an error occurs after the *Method* has returned, a *Fault* flag shall be set and an appropriate *FaultCode* will be returned. The *Fault* (and *FaultCode*) will reset on the next successful Instrument *WriteValue Method* invocation.

Method Declaration

WriteValue (

```
   [in] Value    Float
);
```

Table 18 – Instrument WriteValue Method parameters

Argument	**Description**
Value	Float value *Variable*, that indicates the target state of the *Variable*

Method result codes are defined as part of the *Call Service* (see OPC UA Services Part 4 – Services specification). They are described in Table 93 for ease of reference.

Comments

The *WriteValue Method* will change the value of the *ProcessVariable Variable*. The *Method* will report if any error occurs while writing the value of the *Object*. Table 19 specifies the *AddressSpace* representation for the *WriteInstrumentValueMethod*.

Table 19 – Instrument WriteValue Method AddressSpace Definition

Attribute	**Value**				
BrowseName	Value				
References	**Node Class**	***BrowseName***	**DataType**	**TypeDefinition**	**Modelling Rule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory

5.6 MDISChokeObjectType

5.6.1 Introduction

The following section details the generic *MDISChokeObjectType* structure and defines the properties associated with it. This is in general a vendor and operator independent description, but all users of the *MDISChokeObjectType* can add vendor specific data. The vendor specific data should be defined as part of a subtype of the *MDISChokeObjectType* defined in this document. It is assumed that the subsea system is the *Server* and host of the instance of the *MDISChokeObjectType*. The DCS based system is the *Client* in the system. It is assumed that all interactions with the instance of the *MDISChokeObjectType* are initiated by the *Client* and are directed to the *Server*.

5.6.2 Overview

The *MDISChokeObjectType* is a basic component of any subsea control system. Subsea and surface trees have a choke valve and it is used for regulating the flow volume and with it the back pressure of liquids or gas. This document will address the hydraulic choke valve found in subsea production and water injection trees. Implementation shall ensure adherence to *Mandatory* [M] aspects in order to comply with the MDIS interface standardisation. *Optional* [O] may or may not be implemented within a project. Figure 14 provides an overview of the Choke Object as defined by MDIS. It includes all items that are defined by the *MDISBaseObjectType*. It illustrates that an interlock might have one or more Interlockvariables associated with it, or that they might not have an actual interlock associated.

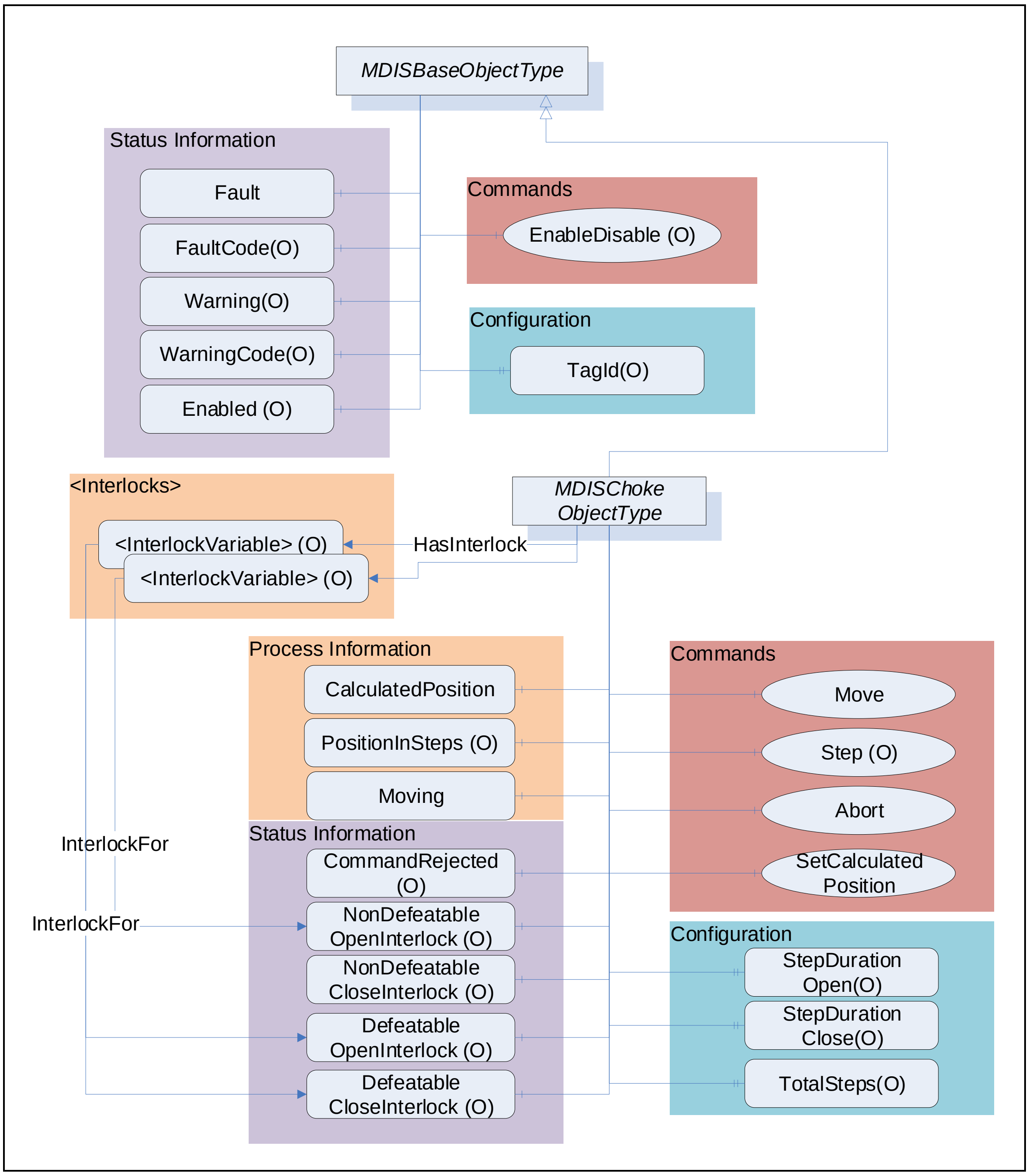

Figure 14 - MDISChokeObjectType

5.6.3 MDISChokeObjectType Definition

Table 20 defines the structure of an *MDISChokeObjectType*. Any vendor specified properties that have been implemented within a project should be documented within a similar format and supplied to the DCS vendor. The addition of vendor specific properties will result in a subtype of the *MDISChokeObjectType*. When an MDISChokeObjectType Instance is disabled the MDISBaseObjectType defaults are followed and only the StepDurationOpen, StepDurationClose and TotalSteps values will be available.

Table 20 - MDISChokeObjectType

Attribute	**Value**					
BrowseName	MDISChokeObjectType					
IsAbstract	False					
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**	**RW**
Subtype of the MDISBaseObjectType						
HasComponent	Variable	CalculatedPosition	Float	BaseDataVariableType	Mandatory	R
HasComponent	Variable	SetCalculatedPositionStatus	SetCalculatedPositionEnum	BaseDataVariableType	Optional	R
HasComponent	Variable	PositionInSteps	Int16	BaseDataVariableType	Optional	R
HasComponent	Variable	Moving	ChokeMoveEnum	BaseDataVariableType	Mandatory	R
HasComponent	Variable	CommandRejected	Boolean	BaseDataVariableType	Optional	R
HasComponent	Method	Move	See 5.6.4		Mandatory	
HasComponent	Method	Step	See 5.6.5		Optional	
HasComponent	Method	Abort	See 5.6.6		Mandatory	
HasComponent	Method	SetCalculatedPosition	See 5.6.7		Mandatory	
HasComponent	Variable	NonDefeatableOpenInterlock	Boolean	BaseDataVariableType	Optional	R
HasComponent	Variable	DefeatableOpenInterlock	Boolean	BaseDataVariableType	Optional	R
HasComponent	Variable	NonDefeatableCloseInterlock	Boolean	BaseDataVariableType	Optional	R
HasComponent	Variable	DefeatableCloseInterlock	Boolean	BaseDataVariableType	Optional	R
HasProperty	Variable	StepDurationOpen	Duration	PropertyType	Optional	R
HasProperty	Variable	StepDurationClose	Duration	PropertyType	Optional	R
HasProperty	Variable	TotalSteps	UInt16	PropertyType	Optional	R
HasInterlock	Variable	<InterlockPlaceholder>		InterlockVariableType	OptionalPlaceholder	

CalculatedPosition – A floating point number that represents the estimated percent open of the choke. This value can be updated using the *SetCalculatedPosition Method*.

SetCalculatedPositionStatus – an enumeration that reflect the status of a SetCalculatedPosition Command. This variable is present if the SetCalculatedPosition command can return asynchronously.

PositionInSteps – An int16 that represents position in steps for the choke.

CommandRejected –– A flag that, if set to True, indicates that the choke has rejected the last command issued to it. The command could be rejected for a number of reasons. Possible reasons for rejecting a command include:

- Loss of subsea communication reported by the SPCS.
- An active interlock.
- The choke is in the disabled state

Enabled – This *Boolean* reflects if the choke is available for control. If it is disabled (FALSE) then the choke will not act on any *Move* command, Step Command or SetCalculatedPosition Command and is not available from a functional point of view. The choke will still report any status information.

Moving – An enumeration indicating the confirmed operation of the choke, (confirmed by SPCS Vendor). Possible status for a choke is moving and stopped.

Move – This *Method* allows an operator to increase or decrease the size of the opening in the choke. This command moves the choke to the percent value provided as part of the command.

Step – This *Method* allows an operator to increase or decrease the size of the opening in the choke. This command moves the choke the number of steps provided as part of the command.

Abort – This *Method* allows an operator to cancel any currently active move or step command.

SetCalculatedPosition – This *Method* is used to calibrate the *CalculatedPosition*. It can only be called when the choke is not moving.

EnableDisable – The choke, when disabled, places a non-defeatable interlock set on *Move* and *Step* functionality, in addition to the functionality described in the *MDISBaseObjectType*.

StepDurationOpen – SPCS open step duration period. This is the time in milliseconds for the choke to open one step.

StepDurationClose – SPCS close step duration period. This is the time in milliseconds for the choke to close one step.

TotalSteps – Total number of steps is the max steps of a choke.

<InterlockPlaceholder> – The number of interlock *Variables* will change based on the project and even choke instance. The *Variables* shall be of *InterlockVariableType* or a subtype of it. They will be referenced by a *HasInterlock Reference* and will contain an *InterlockFor Reference*. *Clients* can use this information to categorise the interlocks appropriately.

The following *Variables* indicate that an interlock is set (TRUE) or is not set (FALSE). The *Variable* shall be the target of an *InterlockFor Reference* from an instance of an *InterlockVariableType* that describes the actual interlock.

NonDefeatableOpenInterlock – The open choke command is interlocked and cannot be overridden.

DefeatableOpenInterlock – The open choke command is interlocked and can be overridden.

NonDefeatableCloseInterlock – The close choke command is interlocked and cannot be overridden.

DefeatableCloseInterlock – The close choke command is interlocked and can be overridden.

5.6.4 Choke Move Method

Move Method is used to adjust the opening size in a choke.

Method Declaration:

Move(

```
[in] Position           Float,
[in] OverrideInterlocks Boolean,
[in] SEM                SEMEnum
);
```

Table 21 – Choke Move Method Arguments

Argument	Description
Position	A number (in percent) indicating the percent open to be moved to when operated.
OverrideInterlocks	Boolean indicating if the open or close command should override any defeatable interlocks
SEM	The selection of which SEM to send the command to.

Method result codes are defined as part of the *Call Service* (see OPC UA Services Part 4 – Services specification). They are described in Table 93 for ease of reference.

Comments:

The *Move Method* initiates a move command. Parameters include the position value in percent, overriding of any interlocks and the SEM selection to use for the command. After receiving the new commanded position, the choke will start to move. The *Method* will complete when the command has been accepted. The move operation may or may not have completed when the *Method* returns. The *Method* returns errors only if the operation cannot be started. The *Client* must monitor the *Moving* Variable to determine when the move operation actually finishes.

If a *Server* does not support an Override of defeatable interlocks, then this parameter will be ignored by the *Server*. If any interlocks are active the appropriate error code is returned.

If a *Server* does not support the selection of SEM, this parameter is ignored.

Table 22 specifies the *AddressSpace* representation for the Choke*MoveMethod*.

Table 22 – Choke Move Method AddressSpace Definition

Attribute	**Value**				
BrowseName	Move				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory

5.6.5 Choke Step Method

Choke Step Method is used to adjust the opening size in a choke.

Method Signature:

```
Step(

[in] Direction           ChokeCommandEnum,
[in] Steps               UInt16,
[in] OverrideInterlocks  Boolean,
[in] SEM                 SEMEnum
);
```

Table 23 – Choke Step Method Arguments

Argument	**Description**
Direction	Enumeration to indicate if an open request or close request is being initiated
Steps	The number of steps to either open or close the choke
OverrideInterlocks	Boolean indicating if the open or close command should override any defeatable interlocks
SEM	The selection of which SEM to send the command to.

Method result codes are defined as part of the *Call Service* (see OPC UA Services Part 4 – Services specification). They are described in Table 93 for ease of reference.

Comments

The choke *Step Method* initiates a move command. Parameters include the direction (open or close), the number of steps to step, overriding of any interlocks and the SEM selection to use for the command. After receiving the command, the choke will start to move. The *Method* will complete when the command has been accepted. The move operation may or may not have completed when the *Method* returns. The *Method* returns errors only if the operation cannot be started. The *Client* must monitor the *Moving Variable* to determine when the *Step* command actually finishes.

If a *Server* does not support an override of defeatable interlocks, then the *OverrideInterlocks* parameter will be ignored by the *Server* and if any interlocks are active the appropriate error code is returned.

If a *Server* does not support the selection of SEM, the *SEM* parameter is ignored.

Table 24 specifies the *AddressSpace* representation for the *ChokeStepMethod*.

Table 24 – Choke Step Method AddressSpace Definition

Attribute	**Value**				
BrowseName	Step				
References	**Node Class**	***BrowseName***	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory

5.6.6 Choke Abort Method

Choke Abort Method is used to cancel any active move command in the *Choke*.

Method Signature

Abort ();

Method result codes are defined as part of the *Call Service* (see OPC UA Services Part 4 – Services specification). They are described in Table 93 for ease of reference.

Comments

The choke *Abort Method* will try to cancel any active choke *Move* or *Step* commands. If no *Move* or *Step* command is in progress, the command will be ignored and return successful. The *Method* will complete when the command has been accepted. The abort operation may or may not have completed when the *Method* returns. The *Method* returns errors only if the operation cannot be started. The *Client* shall monitor the *Moving* and if provided the *CommandRejected Variables* to determine if the abort was successful or failed. Table 25 specifies the *AddressSpace* representation for the choke *Abort Method*.

Table 25 – Choke Abort Method AddressSpace Definition

Attribute	**Value**				
BrowseName	Abort				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**

5.6.7 Choke SetCalculatedPosition Method

SetCalculatedPosition Method is used to synchronise the *CalculatedPosition* to the actual choke position.

Method Signature:

```
SetCalculatedPosition(
[in] CalculatedPosition          Float
);
```

Table 26 – Choke SetCalculatedPosition Method Arguments

Argument	**Description**
CalculatedPosition	A number (in percent)

Method result codes are defined as part of the *Call Service* (see OPC UA Services Part 4 – Services specification). They are described in Table 93 for ease of reference.

Comments:

The *SetCalculatedPosition Method* is used to set the *CalculatedPosition*. It can only be called when the choke is not moving. The parameter is the calculated position. This method may return when the CalculatedPosition has been updated or it may return a status of Completes_Asynchronously. If it returns Completes_Asynchronously the Client will have to monitor the SetCalculatedPostionStatus to determine if an error occurred or the command completed. The *SetCalculatedPositionStatus* will reset on the next successful SetCalculatedPostion *Method* invocation.

Table 27 specifies the *AddressSpace* representation for the *SetCalculatedPosition Method*.

Table 27 – Choke SetCalculatedPosition Method AddressSpace Definition

Attribute	**Value**				
BrowseName	SetCalculatedPosition				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory

5.7 MDISValveObjectType

5.7.1 Introduction

The following section details the generic *MDISValveObjectType* structure and defines the properties associated with it. This is in general a vendor and operator independent description, but all users of this *MDISValveObjectType* can add vendor specific data. The vendor specific data should be defined as part of a subtype of the *MDISValveObjectType* defined in this document. It is required that the subsea system is the *Server* and host of the valve *Object*. The DCS based system is the *Client* in the system. It is required that all interaction with the valve *Object* is initiated by the *Client* and is directed to the *Server*.

5.7.2 Overview

The valve *Object* is a basic component of any subsea control system. Subsea and surface trees have a large variety of valve configurations and combinations of manual and / or actuated (hydraulic or pneumatic) valves. Implementation shall ensure adherence to *Mandatory* [M] aspects in order to comply with the MDIS interface standardisation. *Optional* [O] may or may not be implemented within a project. Figure 15 provides an overview of the *MDISValveObjectType* as defined by MDIS. The figure includes all items inherited from the *MDISBaseObjectType*. It illustrates that an interlock might have one or more Interlockvariables associated with it, or that they might not have an actual interlock associated.

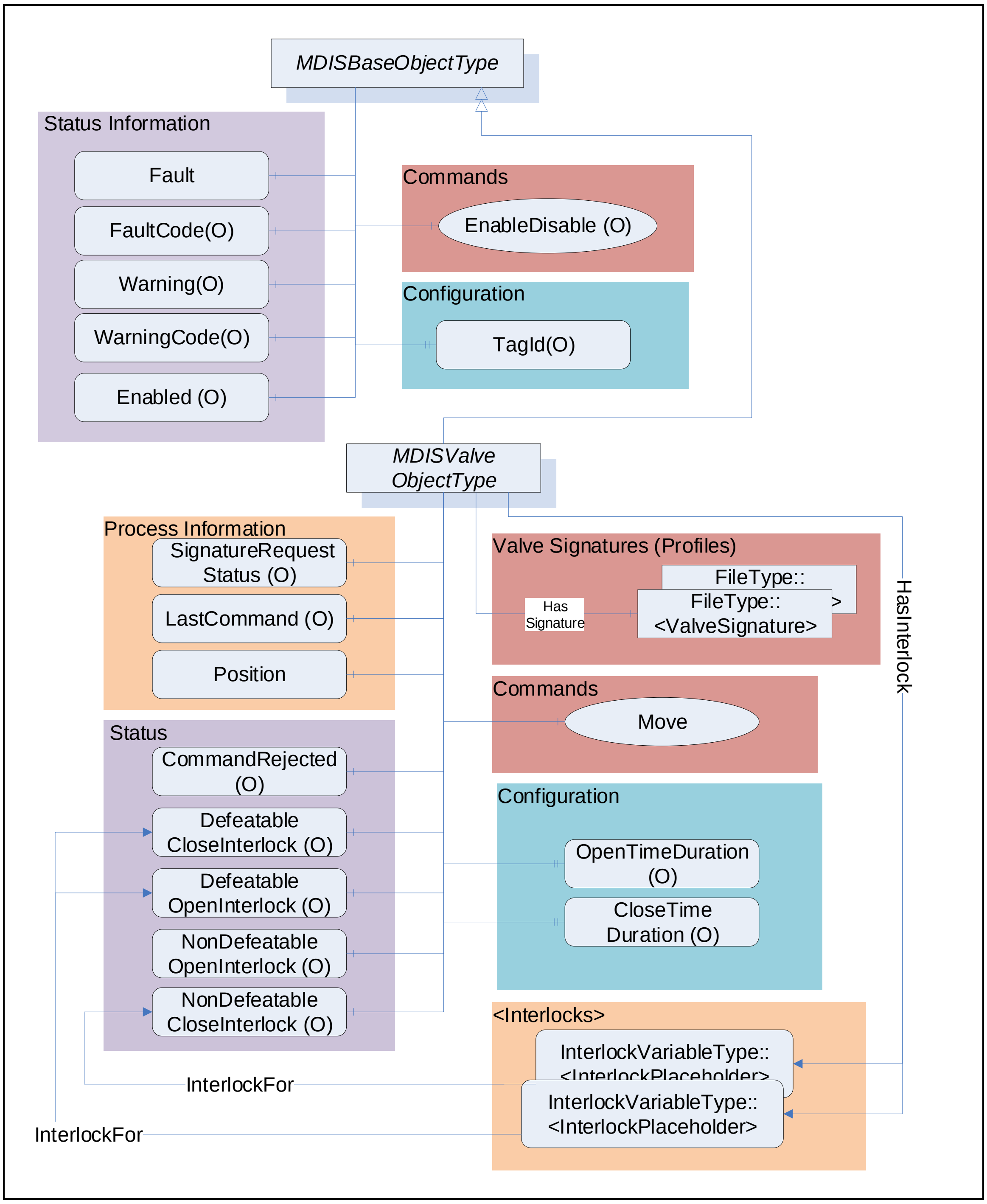

Figure 15 - Valve Object Overview

5.7.3 MDISValveObjectType Definition

Table 28 details the generic properties for the *MDISValveObjectType.* Any vendor specified properties that have been implemented within a project should be documented within a similar format and supplied to the DCS vendor. The addition of vendor specific properties will result in a subtype of the *MDISValveObjectType*. When an MDISValveObjectType Instance is disabled the MDISBaseObjectType defaults are followed and only the OpenTimeDuration and CloseTimeDuration values will be available.

Table 28 - MDISValveObjectType

Attribute	**Value**					
BrowseName	MDISValveObjectType					
IsAbstract	False					
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**	**RW**
Subtype of the MDISBaseObjectType						
HasComponent	Variable	Position	ValvePositionEnum	BaseDataVariableType	Mandatory	R
HasComponent	Variable	CommandRejected	Boolean	BaseDataVariableType	Optional	R
HasComponent	Variable	SignatureRequestStatus	SignatureStatusEnum	BaseDataVariableType	Optional	R
HasComponent	Variable	LastCommand	CommandEnum	BaseDataVariableType	Optional	R
HasComponent	Variable	NonDefeatableOpenInterlock	Boolean	BaseDataVariableType	Optional	R
HasComponent	Variable	DefeatableOpenInterlock	Boolean	BaseDataVariableType	Optional	R
HasComponent	Variable	NonDefeatableCloseInterlock	Boolean	BaseDataVariableType	Optional	R
HasComponent	Variable	DefeatableCloseInterlock	Boolean	BaseDataVariableType	Optional	R
HasComponent	Method	Move	See 5.7.4		Mandatory	
HasInterlock	Variable	<InterlockPlaceholder>		InterlockVariableType	OptionalPlaceholder	R
HasProperty	Variable	OpenTimeDuration	Duration	PropertyType	Optional	R
HasProperty	Variable	CloseTimeDuration	Duration	PropertyType	Optional	R
HasSignature	Object	<ValveSignature>		FileType	Optional	

The following items describe status information from the valve:

- *SignatureRequestStatus* – The collection of data required for a valve signature may take some time after the completion of a *Move* command. The *SignatureRequestStatus Variable* indicates the status of the current signature request (see 7.1.4 for a description of the possible enumerations).

- *LastCommand* – The enumeration reflects the last command sent to the equipment by the SPCS (see 7.1.5 for a description of the possible enumerations).

- *Enabled* – This *Boolean* reflects if the valve is available for control. If it is disabled (FALSE) than the valve will not act on any *Move* command and is not available from a functional point of view. The valve will still report any status information (CommandRejected). One of the cases for a disabled valve is when the valve is placed out of service.

- *Position* – This enumeration provides information about the current position of the valve (see section 7.1.7 for additional details).

- *CommandRejected* – A flag that, if set to True, indicates that the valve has rejected the last command issued to it. The command could be rejected for a number of reasons. Possible reasons for rejecting a command include:

    - Loss of subsea communication reported by the SPCS.
    - An active interlock.
    - The valve is in the disabled state.

For any of the following *Variables*, if they are true, there shall be at least one instance of an *InterlockVariableType* that describes the active interlock.

- *NonDefeatableOpenInterlock* – The open valve command is interlocked and cannot be overridden.
- *DefeatableOpenInterlock* – The open valve command is interlocked and can be overridden.
- *NonDefeatableCloseInterlock* – The close valve command is interlocked and cannot be overridden.
- *DefeatableCloseInterlock* – The close valve command is interlocked and can be overridden.

The following items are related to valve control: These items allow information to flow from the DCS to the SPCS. All commands from the DCS to SPCS may require access controls to ensure that only appropriate personnel initiate them.

- Move – This *Method* causes the valve to open or close (see 5.7.4 for additional details).

<InterlockPlaceholder> –The number of interlock *Variables* will change based on the project and even valve instance. The *Variables* shall be of *InterlockVariableType* or a subtype of it. The interlock contains an *InterlockFor Reference* to one of the previously described flags. *Clients* can use this information to categorise the interlocks appropriately. Figure 16 provides an example of how this interlock *Variable* is used and could be deployed.

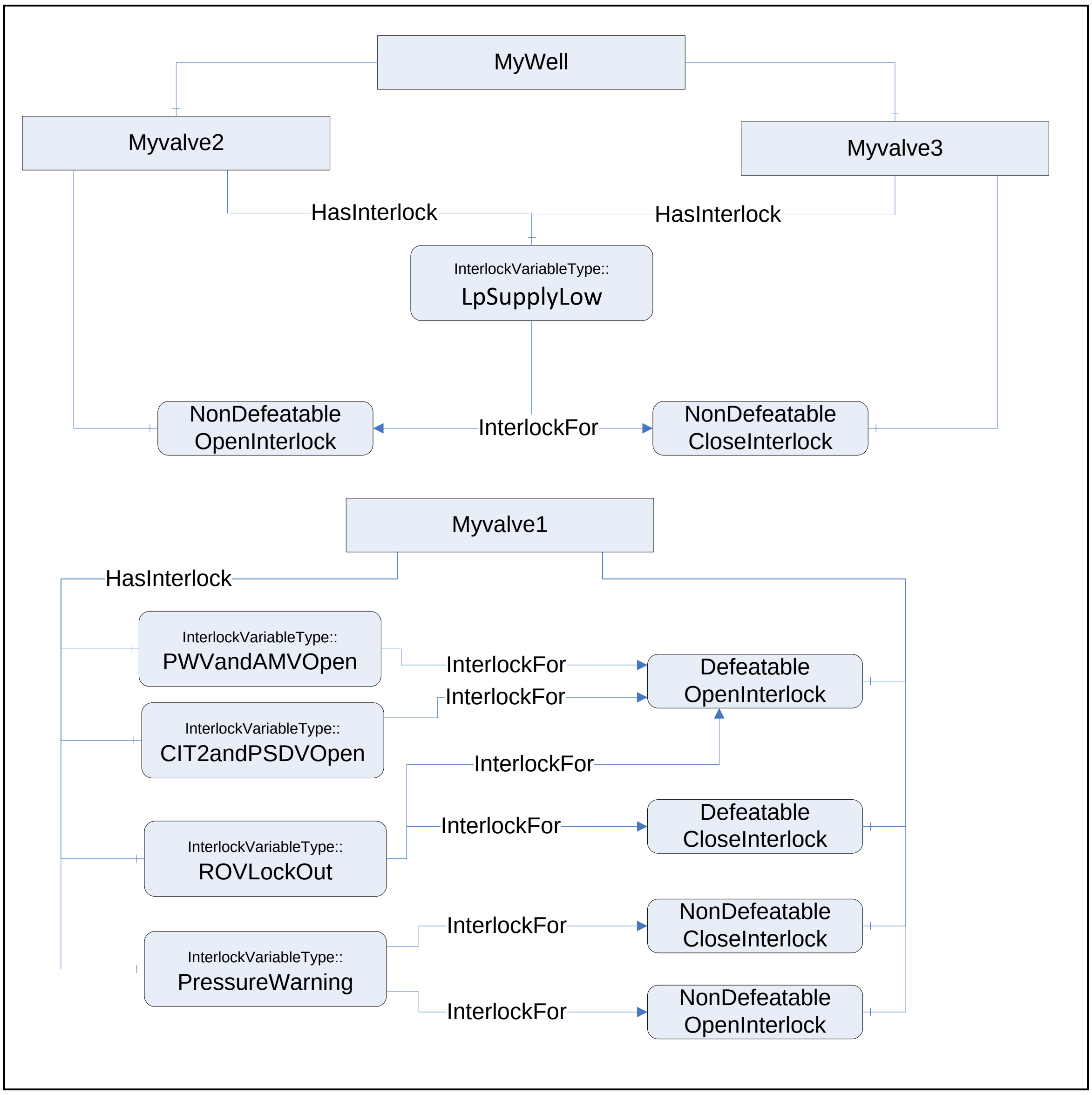

Figure 16 - Interlock example

Note: in OPC UA it is allowed to have a *Node* which is the target of multiple *HasComponent References*, this allows a single interlock to be shared between multiple *Objects*; such as a low pressure interlock that maybe shared by all of the valves and chokes in a system.

The following items describe configuration related items:

- *OpenTimeDuration* – Estimated max time to travel to open position, used for DCS systems to indicate a move fail condition if time is exceeded. Time is provided in milliseconds. The *OpenTimeDuration* can be used for any kind of valve (hydraulic, electric, etc.). [Note: this time is an estimate and *Clients* may need to allow for additional delays in transmitting and receiving any move commands].

- *CloseTimeDuration* - Estimated max time to travel to close position, used for DCS systems to indicate a move fail condition if time is exceeded. Time is provided in milliseconds. The *CloseTimeDuration* can be used for any kind of valve (hydraulic, electric, etc.) [Note: this time is an estimate and *Clients* may need to allow for additional delays in transmitting and receiving any Move commands].

- <*ValveSignature*> - The reference shall point to an instance of *FileType*, where the file contains valve signature information. The name of this object is project or vendor specific, but it should be related to the name of the instance of the *ValveObjectType.* The name shall include a timestamp. The FileType ObjectType (defined in Part 5 – Information Model) includes two properties (illustrated below), for all MDIS instances both of these properties shall have the value False. For additional detail see 5.10.3.

HasProperty	Variable	Writable	Boolean	PropertyType	Mandatory
HasProperty	Variable	UserWritable	Boolean	PropertyType	Mandatory

Vendor specific subtypes of this object type are allowed. They may add additional vendor specific information or may change some *Optional* items to *Mandatory*.

5.7.4 Move Method

Move Method is used to open or close a valve.

Method Declaration:

```
Move(
[in] Direction           CommandEnum,
[in] OverrideInterlocks  Boolean,
[in] SEM                 SEMEnum,
[in] Signature           Boolean,
[in] ShutdownRequest     Boolean
);
```

Table 29 - Move Arguments

Argument	Description
Direction	The enumeration indicates whether the command is to open the valve or to close the valve
OverrideInterlocks	Boolean indicating if the open or close command should override any defeatable interlocks
SEM	The enumeration indicates which SEM to send the command to.
Signature	Boolean indicating if a profile /signature should be generated by this move command request. If the optional *Variable SignatureRequestStatus* is not provided on the *Object*, this parameter is ignored by the *Server*.
ShutdownRequest	Boolean indicates that this command is part of a shutdown sequence.

Method result codes are defined as part of the *Call Service* (see OPC UA Services Part 4 – Services specification). They are described in Table 93 for ease of reference**.**

Comments:

The *Move Method* initiates a move operation. Parameters include opening or closing of the valve, overriding of any interlocks, the SEM selection to use for the command, if a profile / signature is being requested for this move operation or if a shutdown is being requested. The *Method* will complete when the command has been accepted. If the command has not been accepted by the *Server*, then the *Method* returns an error indicating the operation could not be performed. In the case when the *Move* command is accepted by the *Server*, the move operation may or may not be complete at the time the *Method* returns to the *Client*. Therefore, the *Client* must monitor the *Position* of the valve to determine when the *Move* command actually finishes.

The *OverrideInterlocks*, *SEM*, *Signature* and *Shutdown* are optional parameters, but OPC UA *Methods* do not allow for optional parameters. These parameters must always be provided. On a *Server* basis the parameter may be configured to not be utilised. If a *Server* is configured to not utilise a parameter, this is because the functionality is not required.

Table 30 specifies the *AddressSpace* representation for the *Move Method*.

Table 30 - Move Method AddressSpace Definition

Attribute	**Value**				
BrowseName	Move				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasProperty	*Variable*	InputArguments	Argument[]	PropertyType	Mandatory

5.8 MDISAggregateObjectType

5.8.1 Overview

The following tables detail the generic properties for the *MDISAggregateObjectType*; implementation shall ensure adherence to *Mandatory* [M] aspects to comply with the MDIS interface standardisation. *Optional* [O] may or may not be implemented within a project. Figure 17 provides an overview of the *MDISAggregateObjectType* as defined by MDIS. This *Object* is intended to be the base object for all aggregate *ObjectTypes*. See section 9.5 for additional details on data aggregation.

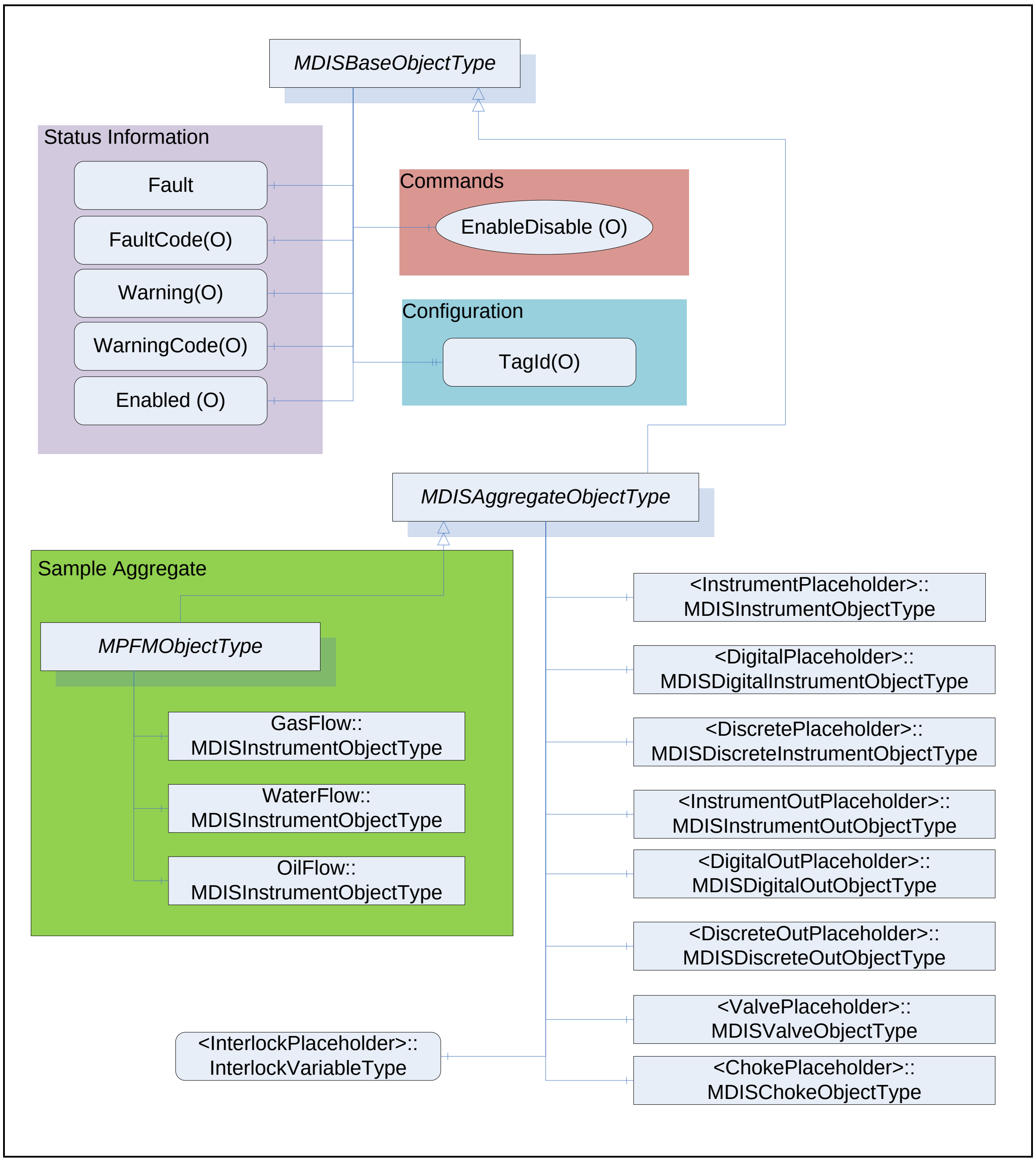
MDISBaseObjectType
Status Information
Fault
FaultCode(O)
Warning(O)
WarningCode(O)
Enabled (O)
Commands
EnableDisable (O)
Configuration
TagId(O)
MDISAggregateObjectType
Sample Aggregate
MPFMObjectType
GasFlow:: MDISInstrumentObjectType
WaterFlow:: MDISInstrumentObjectType
OilFlow:: MDISInstrumentObjectType
<InstrumentPlaceholder>:: MDISInstrumentObjectType
<DigitalPlaceholder>:: MDISDigitalInstrumentObjectType
<DiscretePlaceholder>:: MDISDiscreteInstrumentObjectType
<InstrumentOutPlaceholder>:: MDISInstrumentOutObjectType
<DigitalOutPlaceholder>:: MDISDigitalOutObjectType
<DiscreteOutPlaceholder>:: MDISDiscreteOutObjectType
<ValvePlaceholder>:: MDISValveObjectType
<ChokePlaceholder>:: MDISChokeObjectType
<InterlockPlaceholder>:: InterlockVariableType

Figure 17 – MDISAggregateObjectType

5.8.2 MDISAggregateObjectType Definition

Table 31 defines the structure of an *MDISAggregateObjectType*. The *MDISAggregateObjectType* is a subtype of *MDISBaseObjectType* and requires that all subtypes include, as a minimum, the *Fault* information. All other components are *Optional* and only components that are required by the aggregate are needed.

Table 31 – MDISAggregateObjectType

Attribute	Value					
BrowseName	MDISAggregateObjectType					
IsAbstract	True					
References	**Node Class**	**BrowseName**	**Data Type**	**TypeDefinition**	**Modelling Rule**	**RW**
Subtype of the MDISBaseObjectType defined in section 5.1.1						
HasComponent	Object	<InstrumentPlaceholder>		MDISInstrumentObjectType	Optional Placeholder	
HasComponent	Object	<InstrumentOutPlaceholder>		MDISInstrumentOutObjectType	Optional Placeholder	
HasComponent	Object	<DigitalInstrumentPlaceholder>		MDISDigitalInstrumentObjectType	Optional Placeholder	
HasComponent	Object	<DiscreteInstrumentPlaceholder>		MDISDiscreteInstrumentObjectType	Optional Placeholder	
HasComponent	Object	<DigitalOutPlaceholder>		MDISDigitalOutObjectType	Optional Placeholder	
HasComponent	Object	<DiscreteOutPlaceholder>		MDISDiscreteOutObjectType	Optional Placeholder	
HasComponent	Object	<ValvePlaceholder>		MDISValveObjectType	Optional Placeholder	
HasComponent	Object	<ChokePlaceholder>		MDISChokeObjectType	Optional Placeholder	
HasComponent	Variable	<InterlockPlaceholder>		InterlockVariableType	Optional Placeholder	

The *MDISAggregateObjectType* is an abstract *ObjectType*; instances of this *ObjectType* cannot be created. *Object* instances can only be created of subtypes of this *ObjectType*. In OPC UA it is legal to add additional *Object* or *Variable Reference*(s) to an instance of an *Object*, (i.e. add *Variable* or *Object* to an instance that are not defined in the type), but in MDIS we are restricting this in that a *Client* is not required to process or handle any *Objects* or *Variables* that are not part of a type.

The subtypes of *MDISAggregateObjectType* are allowed to include other subtypes of *MDISAggregateObjectType*. For example, a Well that is defined as a subtype of *MDISAggregateObjectType* might include an MPFMAggregateObjectType which is also a subtype of *MDISAggregateObjectType*.

<InstrumentPlaceholder> denotes that a subtype of this *ObjectType* may define any number of *Objects* of this type as part of a subtype. Each object instance shall have a unique *BrowseName* and must be of *MDISInstrumentObjectType*.

<InstrumentOutPlaceholder> denotes that a subtype of this *ObjectType* may define any number of *Objects* of this type as part of a subtype. Each object instance shall have a unique *BrowseName* and must be of *MDISInstrumentOutObjectType.*

<DigitalInstrumentPlaceholder> denotes that a subtype of this *ObjectType* may define any number of *Objects* of this type as part of a subtype. Each object instance shall have a unique *BrowseName* and must be of *MDISDigitalInstrumentObjectType*.

<DiscreteInstrumentPlaceholder> denotes that a subtype of this *ObjectType* may define any number of *Objects* of this type as part of a subtype. Each object instance shall have a unique *BrowseName* and must be of *MDISDiscreteInstrumentObjectType*.

<DigitalOutPlaceholder> denotes that a subtype of this *ObjectType* may define any number of *Objects* of this type as part of a subtype. Each object instance shall have a unique *BrowseName* and must be of *MDISDigitalOutObjectType*.

<DiscreteOutPlaceholder> denotes that a subtype of this *ObjectType* may define any number of *Objects* of this type as part of a sub type. Each object instance shall have a unique *BrowseName* and must be of *MDISDiscreteOutObjectType*.

<ValvePlaceholder> denotes that a subtype of this *ObjectType* may define any number of *Objects* of this type as part of a subtype. Each object instance shall have a unique *BrowseName* and must be of *MDISValveObjectType* .

<ChokePlaceholder> denotes that a subtype of this *ObjectType* may define any number of *Objects* of this type as part of a subtype. Each object instance shall have a unique *BrowseName* and must be of *MDISChokeObjectType*.

<InterlockPlaceholder> denotes that a subtype of this *VariableType* may define any number of *Variables* of this type as part of a subtype. Each *Variable* instance shall have a unique *BrowseName* and must be of *InterlockVariableType.*

5.9 MDISTimeSyncObjectType

5.9.1 Introduction

In some systems, a MDIS *Server* might be on an isolated network, one in which there is no NTP or other source of time synchronization signals. In these systems, the MDIS *Server* would need to be able to obtain a minimum time synchronization signal from the *Client*. This *ObjectType* and associated method is designed to allow this minimum time synchronization. It should only be used if a better time synchronization system, such as NTP, is not available. The accuracy of this type of time synchronization could be in the range of seconds.

If this *Object* is supported, a single instance of this *Object* shall exist on a *Server*. The *Object* shall have a name *TimeSynchronization* and the instance shall exist as part of the *MDISInformationObject*.

5.9.2 Overview

The following section details the *MDISTimeSyncObjectType*. This optional type allows a *Client* to provide time synchronization information to a *Server*. The *Client* can call the *SetTime Method* periodically to ensure the *Server* time does not drift. This Method does not return until the time on the Server has been updated or an error is returned.

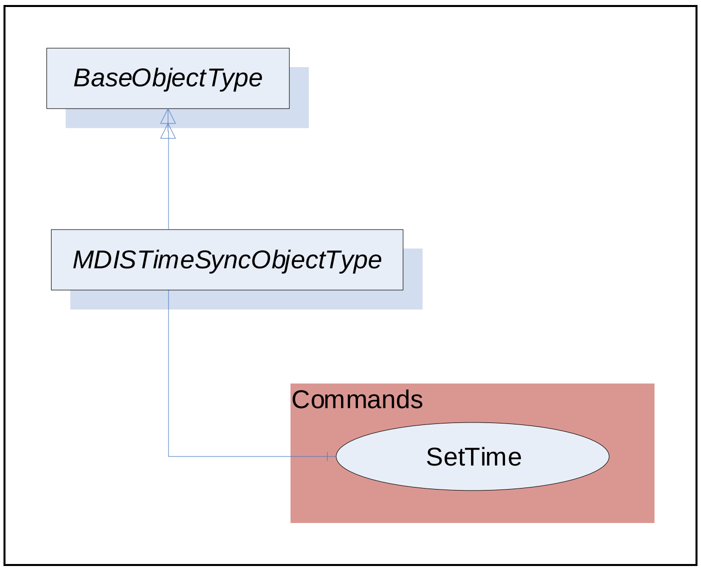

Figure 18 - MDISTimeSyncObjectType

5.9.3 MDISTimeSyncObjectType Definition

Table 32 defines the structure of an *MDISTimeSyncObjectType*.

The current time on the *Server* is available as part of the *ServerStatus* provided by all OPC UA *Servers.*

Table 32 - MDISTimeSyncObjectType

Attribute	Value					
BrowseName	MDISTimeSyncObjectType					
IsAbstract	False					
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**	**R W**
Subtype of the BaseObjectType						
HasComponent	Method	SetTime	SetTime Method is defined in 5.9.4		Mandatory	

SetTime – This method allows a *Client* to set time on the *Server*.

5.9.4 SetTime Method

SetTime Method (defined in Table 33) is used to set the time on the *Server*. If an error occurs this *Method* shall return an error code indicating the failure to set the time (see Table 35).

Method Declaration

SetTime (

```
   [in] TargetTime    UtcTime
);
```

Table 33 - SetTime Method parameters

Argument	Description
TargetTime	The UTC Time that the *Server* shall use to update its internal clock.

Method result codes are defined as part of the *Call Service* (see OPC UA Services Part 4 – Services specification). They are described in Table 93 for ease of reference.

Comments

The *SetTime Method* will change the time on the *Server*. Table 10 specifies the *AddressSpace* representation for the *SetTime Method*.

Table 34 - SetTime Method AddressSpace Definition

Attribute	Value				
BrowseName	SetTime				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory

Table 35 – SetTime Result Codes

Symbolic Id	Description
Bad_InvalidTimestamp	The timestamp is outside the range allowed by the *Server* or an error occurred setting the Server time.
	.

5.10 MDISInformationObjectType

5.10.1 Introduction

MDIS defines flags, general status information and other optional data collections that need to exist in a standard manner on MDIS *Servers* to allow for easier interoperation between *Clients* and *Servers*. This information is provided via the MDISInformationObjectType.

5.10.2 Overview

The following section details the *MDISInformationObjectType*. This *ObjectType* defines a standard structure for organization of some basic *Server* information. An instance of this object is required for all versions of MDIS *Servers* 1.2 and greater. An instance of this *ObjectType* shall exist under the *Objects* folder on all OPC UA MDIS *Servers*. *Clients* are required to be able to handle *Servers* which do not contain this object. The *Client* shall assume any *Server* that does not contain an instance of this *ObjectType* is a version 1.1 or 1.0 MDIS *Server*.

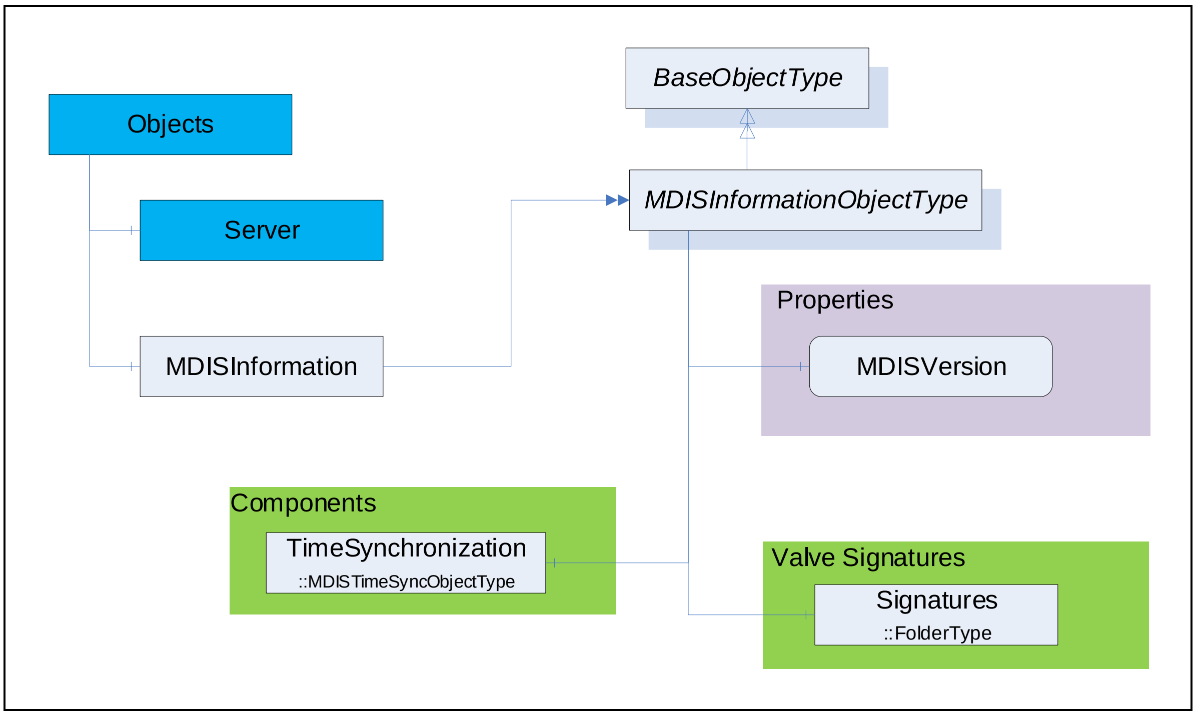

Figure 19 - MDISInformationObjectType

5.10.3 MDISInformationObjectType Definition

Table 7 defines the structure of an *MDISInformationObjectType*.

Table 36 – MDISInformationObjectType

Attribute	**Value**					
BrowseName	MDISInformationObjectType					
IsAbstract	False					
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**	**R W**
Subtype of the BaseObjectType (see Part 5 – Information Model)						
HasComponent	Object	TimeSynchronization		MDISTimeSyncObjectType	Optional	
HasComponent	Object	Signatures		FolderType	Optional	
HasComponent	Variable	MDISVersion	MDISVersionData Type	MDISVersionVariableType	Mandatory	R

TimeSynchronization – this is an instance of the *MDISTimeSynchronizationType* object, that allows a *Client* to provide time information to a *Server* if required. See 5.1.11 for additional details.

Signatures is a folder that contains all of the currently available signatures (profiles). The individual signature(s) are stored as *FileObjects* and the format of the file(s) is vendor specific.

MDISVersion provides information about the version of the MDIS specification that is implemented by the *Server*. This is provided to assist the *Client* in determining what should be available and can also be used for automated testing of the *Server.*

6 MDIS VariableTypes

6.1 InterlockVariableType Definition

This *VariableType* is a Boolean indicating if the interlock is active. The instance of a *Variable* of *InterlockVariableType* shall have a descriptive name (display name) and include a description. The description shall describe the type of interlock. On a project basis an optional summary that details active interlocks may be implemented to support understanding of the open / close interlock status within the *Object*. For a given instance of an *Object*, any number of referenced interlocks may exist, where each instance would represent a specific interlock. The number of interlocks is determined at configuration time of the *Object* and could be different for each *Object* in a given well. Each interlock variable shall contain an *InterlockFor* reference to at least one of the following:

- *NonDefeatableOpenInterlock*,
- *DefeatableOpenInterlock*,
- *NonDefeatableCloseInterlock*,
- *DefeatableCloseInterlock*.

A given instance of an *InterlockVariableType* might be referenced by more than one valve *Object* or choke *Object*, but for all objects that reference a given instance of an *InterlockVariableType* there must be at least one *InterlockFor* reference.

Table 37 - InterlockVariableType

Attribute	**Value**					
BrowseName	InterlockVariableType					
IsAbstract	False					
ValueRank	-1 (-1 = Scalar)					
DataType	Boolean					
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**	**RW**
Subtype of the BaseDataVariableType defined in OPC UA Part 5						

6.2 MDISVersionVariableType Definition

This *VariableType* defines a standard representation of the version information that is related the MDIS Specification.

Table 38 - MDISVersionVariableType

Attribute	**Value**					
BrowseName	MDISVersionVariableType					
IsAbstract	False					
ValueRank	-1 (-1 = Scalar)					
DataType	MDISVersionDataType					
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**	**RW**
Subtype of the BaseDataVariableType defined in OPC UA Part 5						
HasProperty	Variable	MajorVersion	Byte	BaseDataVariableType	Mandatory	R
HasProperty	Variable	MinorVersion	Byte	BaseDataVariableType	Mandatory	R
HasProperty	Variable	Build	Byte	BaseDataVariableType	Mandatory	R

MajorVersion number is the major version of the MDIS OPC UA companion specification. For example for version 1.2 of the MDIS Specification this will be a 1.

MinorVersion number is the minor version of the MDIS OPC UA companion specification. For example, for version 1.2 of the MDIS Specification this will be a 2.

Build number is additional version information that can be associated with minor updates, patches that could be provided by the MDIS organization related to the MDIS Information model NodeSet file. Typically, this will be 0.

7 MDIS DataTypes

7.1 Enumerations

7.1.1 ChokeMoveEnum

Table 39 defines the valid states for the *ChokeMoveEnum*.

Table 39 - ChokeMoveEnum

Name	Description
Moving_1	The choke is currently moving (in progress)
Stopped_2	The move has stopped

7.1.2 ChokeCommandEnum

Table 40 defines the valid states for the *ChokeCommandEnum*.

Table 40 - ChokeCommandEnum

Name	Description
Close_1	The command to the Choke is Close
Open_2	The command to the Choke is Open

7.1.3 SetCalculatedPositionEnum

Table 40 defines the valid states for the *SetCalculatedPositionEnum*.

Table 41 - SetCalculatedPositionEnum

Name	Description
Initial_0	no command(initial state)
Inprogress_1	command in progress
Complete_2	command completed
Fault_4	command fault

7.1.4 SignatureStatusEnum

Table 42 defines the valid states for the *SignatureStatusEnum*.

Table 42 - SignatureStatusEnum

Name	Description
NotAvailable_1	The profile / signature is not available (in progress)
Completed_2	The profile / signature request has completed
Failed_4	The profile / signature request has failed

Retrieval of a signature is outside of the scope of this interface. A NotAvailable_1 indicates that the current profile / signature request is not available.

7.1.5 CommandEnum

Table 43 defines the valid states for the *CommandEnum*. This might not be the actual state of the valve, it is just the last command sent to the valve.

Table 43 - CommandEnum

Name	Description
Close_1	The last command to the valve was Close
Open_2	The last command to the valve was Open
None_4	No known command has been sent to the valve. The initial setting on start-up of a server.

7.1.6 SEMEnum

Table 44 defines the valid states for the *SEMEnum*.

Table 44 - SEMEnum

Name	Description
SEM_A_1	Valve move command selection SEM A
SEM_B_2	Valve move command selection SEM B
Auto_4	Subsea equipment vendor decides how to send the command. In some cases, this would be both SEMs, in others it would mean a subsea system's choice of a SEM.

7.1.7 ValvePositionEnum

Table 45 defines the valid states for the *ValvePositionEnum*.

Table 45 - ValvePositionEnum

Name	Description
Close_1	The Valve is Closed
Open_2	The Valve is Open
Moving_4	The Valve is Moving
Unknown_8	The Valve is in an unknown state. This value can be used when a subsea vendor does not have any last command information and does not know the state of the valve.

7.2 Structures

7.2.1 MDISVersionDataType

The *MDISVersionDataType* provides a single structure that provides all of the version information for the *Server*. There is a corresponding variable structure that provides each of the elements.

This *VariableType* is defined in 6.2.

Table 46 – MDISVersionDataType Structure

Name	Type	Description
MDISVersionDataType	Structure	Information that describes MDIS Specification version.
MajorVersion	Byte	The Major Version number from the specification
MinorVersion	Byte	The minor version number from the specification
Build	Byte	The BuildNumber associated – typically always 0

8 MDIS ReferenceTypes

8.1 HasInterlock ReferenceType

The *HasInterlock ReferenceType* is a concrete *ReferenceType* that can be used directly. It is a subtype of the *HasComponent ReferenceType*.

The semantic is a part-of relationship. Figure 16 provides an illustration of how this *ReferenceType* is used. Multiple *SourceNodes* can reference the same *TargetNode* and a *SourceNode* can reference multiple *TargetNodes*.

Like all other *ReferenceTypes*, this *ReferenceType* does not specify anything about the ownership of the parts, although it represents a part-of relationship semantic. That is, it is not specified if the *TargetNode* of a *Reference* of the *HasInterlock ReferenceType* is deleted when the *SourceNode* is deleted.

The *TargetNode* of this *ReferenceType* shall be the *InterlockVariableType* or a *Variable* of *InterlockVariableType* or a subtype of *InterlockVariableType*.

The *SourceNode* shall be an instance of *MDISChokeObjectType* or *MDISValveObjectType* or a subtype of *MDISAggregateObjectType* that is being used for aggregation or one of these *ObjectTypes*.

Table 47 - HasInterlock Reference

Attributes	**Value**		
BrowseName	HasInterlock		
InverseName	InterlockOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

8.2 InterlockFor

The *InterlockFor ReferenceType* is a concrete *ReferenceType* and can be used directly. It is a subtype of *NonHierarchicalReferences.* The inverse reference shall also exist for all uses of this *ReferenceType.*

The semantic of this *ReferenceType* is to relate *InterlockVariableType Nodes* to the *Variable Node* that represents an interlock flag.

The *SourceNode* of this *ReferenceType* shall be an instance of *InterlockVariableType*.

The *TargetNode* of this *ReferenceType* shall be a *Variable* that is one of *NonDefeatableOpenInterlock*, *DefeatableOpenInterlock*, *NonDefeatableCloseInterlock* or *DefeatableCloseInterlock*. The *Variable* shall be part of an *MDISValveObjectType* or *MDISChokeObjectType* or *MDISAggregateObjectType*, a subtype of one of these types or an instance of one of these types.

Table 48 - InterlockFor Reference

Attributes	**Value**		
BrowseName	InterlockFor		
InverseName	HasInterlockInformation		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	***BrowseName***	**Comment**

8.3 HasSignature ReferenceType

The *HasSignature ReferenceType* is a concrete *ReferenceType* that can be used directly. It is a subtype of the *HasComponent ReferenceType*.

The semantic is a part-of relationship. Figure 20 provides an illustration of how this *ReferenceType* is used. A TargetNode can be referenced by a single SourceNode and a *SourceNode* can reference multiple *TargetNodes*.

Like all other *ReferenceTypes*, this *ReferenceType* does not specify anything about the ownership of the parts, although it represents a part-of relationship semantic. That is, it is not specified if the *TargetNode* of a *Reference* of the *HasSignature ReferenceType* is deleted when the *SourceNode* is deleted.

The *TargetNode* of this *ReferenceType* shall be an instance of *FileType*. The *FileType* instance shall contain valve signature information.

The *SourceNode* shall be an instance of *MDISValveObjectType* or a subtype of *MDISValveObjectType*. It might also be a *MDISChokeObjectType* or any other MDIS *ObjectType* that might need to report a signature.

Table 49 - HasSignature Reference

Attributes	**Value**		
BrowseName	HasSignature		
InverseName	SignatureOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

9 MDIS AddressSpace Information

9.1 Introduction

This section defines information related to Instance Objects, value handling, *UANodeSet*, and customised *Objects*

9.2 Instance AddressSpace

An MDIS *Server* shall include an instance of the *MDISInformationObjectType* under the *Objects* folder.

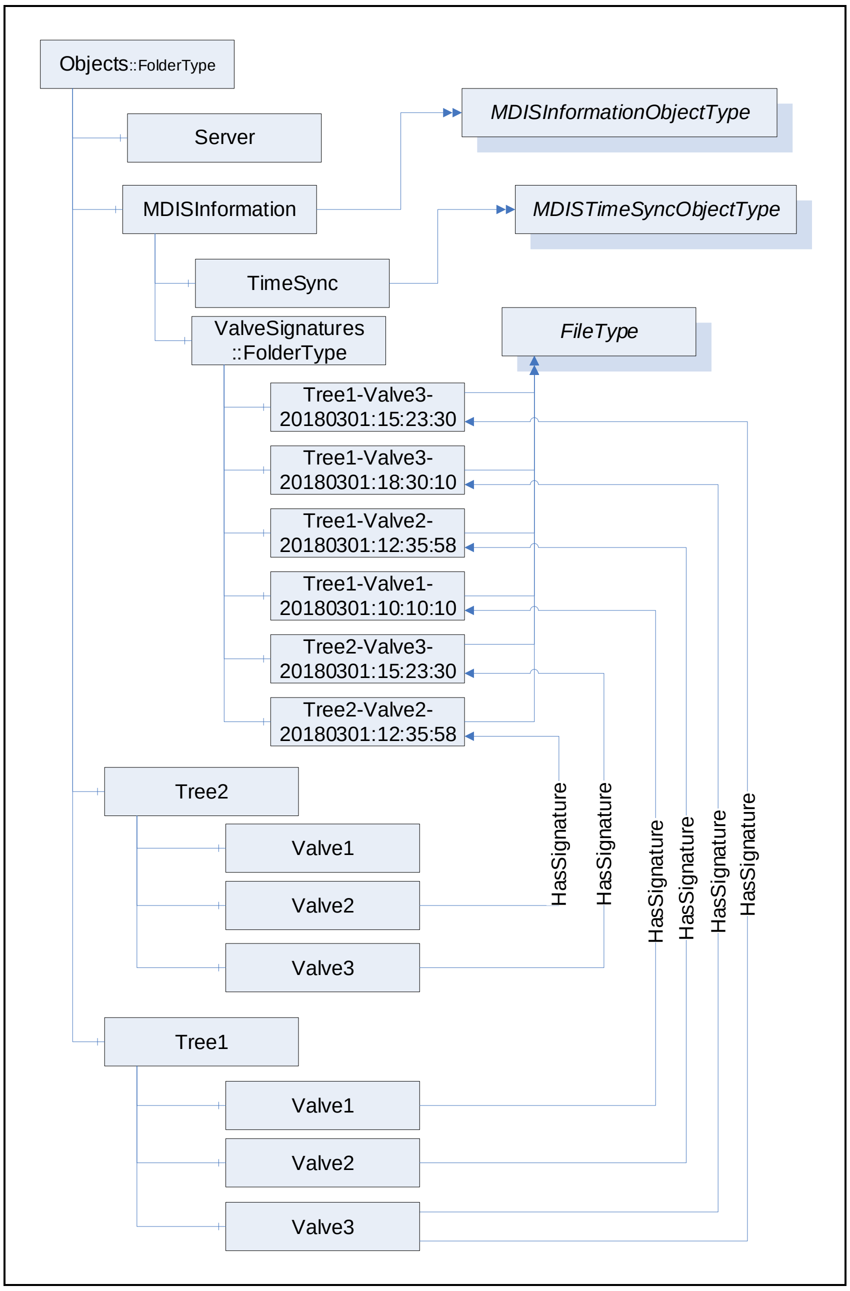
Objects::FolderType
Server
MDISInformationObjectType
MDISInformation
MDISTimeSyncObjectType
TimeSync
ValveSignatures ::FolderType
FileType
Tree1-Valve3-20180301:15:23:30
Tree1-Valve3-20180301:18:30:10
Tree1-Valve2-20180301:12:35:58
Tree1-Valve1-20180301:10:10:10
Tree2-Valve3-20180301:15:23:30
Tree2-Valve2-20180301:12:35:58
Tree2
Valve1
Valve2
Valve3
Tree1
Valve1
Valve2
Valve3
HasSignature
HasSignature
HasSignature
HasSignature
HasSignature
HasSignature

Figure 20 - MDIS Instance illustration

Any other structure under in the MDIS Server is vendor or project specific. Figure 20 provides an illustration of one possible instance structure, it is not the only available structure. Several of the items are optional and might not appear. The valve signature objects illustrate that valves might include multiple signature for a given valve, no signatures for a given valve and that the single folder might contain signature from multiple trees. The naming use is project / vendor specific but might have to handle all of these cases. Signatures might also exist for other objects such as chokes.

The MDISInformation node is formally defined in **Table 50**.

Table 50 – MDISInformation definition

Attribute	**Value**		
BrowseName	MDISInformation		
References	**NodeClass**	**BrowseName**	**TypeDefinition**
OrganizedBy by the Objects Folder defined in Part 5 – Information Model			
HasTypeDefinition	ObjectType	BaseObjectType	

9.3 Value reporting

Data overflow at the SPCS shall be detected by the *Client*. This shall be performed through the use of a queue size greater than 1 on subscriptions and monitoring of the overflow bit in all received values. If an overflow occurs the *Client* shall generate some notification, but the exact notification is *Client* vendor specific.

9.4 UANodeSet Development

In OPC UA an *AddressSpace* is separated into *Namespaces*, where each *Namespace* defines the owner of the address space contained in it. The MDIS *Namespace* shall not be modified in any manner, it is owned by MDIS. Any implementation (custom types / instances) shall be in a separate *Namespace* from the MDIS *Namespace*.

Any custom types or subtypes of the standard MDIS Types, as defined in Section 9.5.2 should be separated from instances via *Namespaces*. This enables custom types to be shared across DCS vendors and SPCS vendors during early phases of project development and allows the DCS vendors to begin implementation of the custom types within the DCS software. It also allows reuse of types between projects.

When *UANodeSet* files are passed from the SPCS vendors to the DCS vendors, the whole address space is to be exchanged (i.e. the exchanged *UANodeSet* files shall include the MDIS *Namespace* and any custom *Namespaces*). A *Namespace* can be in a single *UANodeSet* file and multiple files are provided or the entire address space can be in a single *UANodeSet* file. Having all *Namespaces* shall enable the DCS vendors to understand any forward *References* as forward *References* can create ambiguity within developed *UANodeSet* files.

The length of the namespace URLs used for defining *Namespaces* should be less than 128 characters. This is to enable smaller devices like controllers to be able to store *Namespaces*. *NamespaceUri* usually reference back to the company that is the owner of the *Namespace*.

The *NodeIds* should be integers. If using strings, the length of the *NodeIds* shall be less than 32 characters. This shall enable smaller devices like controllers to process *NodeIds* efficiently.

NodeIds shall be fixed. When a *Server* restarts *NodeIds* shall not be changed. *NodeIds* should not be reused on a project. Any change in *NodeIds* shall require *Clients* to reconfigure the connection based on new generated *UANodeSet* files. Once a *Nodeset* file is delivered to a DCS vendor from the SPCS vendor the *NodeIds* in the file are fixed. Future NodeSet files might add or delete nodes. But in any future *Nodeset* file shall not change the Nodeid of any Objects that have been delivered in the prior NodeSet file.

For custom types the instance hierarchy shall be represented via *HasComponent* (standard OPC UA *Reference*) or a custom *ReferenceType* that has been subtyped from *HasComponent* or *Organizes References* only.

9.5 Object Development

9.5.1 Introduction

This section provides guidance on creation of modified and composite objects for use in compliance with MDIS standards. Standard MDIS *Objects* defined in Section 5 shall be used wherever possible. In the event that standard *Objects* cannot represent a given subsea component or piece of equipment, a new *Object* shall be developed by collection, aggregation or extension of existing *Objects*. Use of collection, aggregation or extension to model subsea equipment is dependent on vendor implementations.

9.5.2 Object Collection, Aggregation and Extension Definition

Object collection shall be implemented using folders or *Objects* derived from *MDISAggregateObjectType* and should be used to group components or equipment that are physically grouped together in a common structure. Several examples of where object collection could be utilised would include modelling Subsea Electronics Modules, Electronic Power Units and Wells.

Object aggregation shall be implemented by creating a new type or subtype and should be used to group relevant objects to represent a complex piece of equipment that cannot be represented by a single MDIS *Object*. Specific examples of where *Object* aggregation could be used include modelling Multiphase Flow Meters (MPFMs) or Chemical Injection Metering Valves (CIMVs). Aggregated *Objects* have specific rules, defined below, to allow *Clients* to be able to discover them and easily support them.

Object extension shall be implemented by creating a subtype of an existing MDIS *Object* and should be used when an existing MDIS *Object*, such as a valve *Object*, has additional information or functionality that needs to be represented. Extension of *Objects* applies to all models defined in section 5 (i.e. *MDISDiscreteInstrumentObjectType*, *MDISDigitalInstrumentObjectType*, *MDISInstrumentObjectType*, *MDISValveObjectType*, *MDISChokeObjectType* or any subtype of these types). Subtyping *MDISBaseObjectType* is not allowed.

Rules for developing aggregated and extended objects are provided in Table 51. To minimise variability, when aggregating or extending objects only instances of the following *VariableTypes* and *ObjectTypes* shall be used:

- Standard MDIS *Objects* defined in section 5.
- *BaseDataVariableType*
- *DiscreteItemType* (Variable)
- *AnalogItemType* (Variable)

Table 51 - Aggregation and Extension Decision Matrix

Description	Aggregation	Extension
Require new *NodeId* on TypeDefinition level	Yes	Yes
0) Define new *ObjectType* with any non-MDIS parent	No	No
1) Define new *ObjectType* with any MDIS parent (subtyping MDIS *ObjectTypes* - *MDISValveObjectType*, *MDISChokeObjectType*, *MDISDigitalInstrumentObjectType*, *MDISInstrumentObjectType*, *MDISDiscreteInstrumentObjectType* or any subtype of these types)	No	Yes
2) Define new *ObjectType* with *MDISAggregateObjectType* as parent	Yes	No

The operations described in Table 52 are valid operations in a generic OPC UA *Server*, but for an MDIS *Server* they are restricted as described in the table. All changes shall be based on type changes. Instance specific changes are not allowed.

Table 52 - General rules that apply to existing MDIS types

Description	Aggregation	Extension
Add MDIS *Object* instance to an instance of an existing MDIS *ObjectType* (or subtype).	No	No
Add non-MDIS *Object* instance to existing MDIS *ObjectType* (or subtype).	No	No
Add non-MDIS *Object* instance to an instance of an existing MDIS *ObjectType* (or subtype).	No	No
Add *Method* to an existing MDIS *ObjectType* (or subtype).	No	No
Add a *Variable* instance to an instance of an existing MDIS *ObjectType* (or subtype).	No	No
Add a *Variable* instance to an existing MDIS *ObjectType*.	No	No
Base MDIS *Object's* compliance to MDIS OPC UA specs shall be demonstrable (CTT).	Yes	Yes

When extending an existing *ObjectType*, the rules described in

Table 53 apply to the MDIS *Objects* defined in Section 5. These rules apply to *MDISValveObjectType*, *MDISChokeObjectType*, *MDISInstrumentObjectType*, *MDISDigitalInstrumentObjectType*, *MDISDiscreteInstrumentObjectType* or any subtype of these types; they do not apply to the *MDISBaseObjectType* and the *MDISAggregateObjectType*. The

MDISBaseObjectType cannot be extended or subtyped by a vendor or project. Only the MDIS working group can extend the *MDISBaseObjectType* or create new a subtype of it. The *MDISAggregateObjectType* has its own set of rules. Extending existing *ObjectTypes* are restricted to limit available additions to allow *Clients* to pick up the new types without requiring coding changes to the *Client*.

Table 53 - Rules for subtypes

Description	Extension
Add a *BaseDataVariableType* instance to the newly create subtype of an MDIS *ObjectType* (*MDISValveObjectType*, *MDISChokeObjectType*, *MDISDigitalInstrumentObjectType*, *MDISDiscreteInstrumentObjectType*, *MDISInstrumentObjectType* or any subtype of these types), with a *DataType* of one of the OPC *BaseDataType* excluding structure.	Yes
Add an instance of an *AnalogItemType* to the newly created subtype of an MDIS *ObjectType*.	Yes
Add an instance of a *DiscreteItemType* to the newly created subtype of an MDIS *ObjectType*.	Yes
Change *ModellingRule* from *Optional* -> *Mandatory* for any of existing properties or components.	Yes

Table 54 describes rules for creating an aggregate object. They only apply as described.

Table 54 - Aggregation Related Rules

Description	Aggregate
Add MDIS *Object InstanceDeclaration* to the newly created subtype of *MDISAggregateObjectType*	Yes
Add a MDIS *Variable* instance to the newly created sub type of the *MDISAggregateObjectType*	Yes
Add a MDIS defined *Reference* to the newly created subtype of *MDISAggregateObjectType*.	Yes

Clients are expected to be able to handle new aggregate *Objects*, even if the *Client* only lists them as separate MDIS *Objects* and *Variables* of which they are composed. *Clients* are not required to handle an extension *Object's* additional *Variables*, but they are required to support instances of the extension object. *Clients* that are unable to handle the extension *Object's* additional *Variables* shall treat the extension object as an instance of the parent standard MDIS *ObjectType*. It is expected that some projects will require supporting the additional *Variables* defined in an extension *Object*.

9.5.3 Object Creation

ObjectTypes that are not an aggregated, extended or collection *ObjectType* can only be defined by the MDIS organisation. If a vendor (or group of vendors) determines that an *ObjectType* might be a candidate for creation as a new *ObjectType*, the proposed *ObjectType* needs to be submitted to or proposed to the MDIS organisation. If the MDIS organisation determines that a new *ObjectType* is to be created, it will be incorporated into the MDIS OPC UA Companion Specification. These new *Objects* shall be reviewed and validated by the MDIS organisation members.

9.5.4 Object Aggregation Example

The following is an example for creating an aggregated model of a simple CIMV Object. A generic model of a CIMV could have the following items:

- Valve Position Target (32 bit, float) – Read / Write
- Valve Flow Target (32 bit, float) – Read / Write
- Valve Status (8 bit, word) – Read Only
- Valve Device State (8 bit, word) – Read Only

The following figure illustrates the resulting *TypeDefinition* with example *Object* instances. It includes two instances of *MDISDiscreteInstrumentObjectType Objects* that represent the Valve Status and the Valve Device State, two instances of *MDISInstrumentOutObjectType Objects* that provide the ability to set the Position and Flow target values and two instances of *MDISInstrumentObjectType Objects* that provide feedback on the set Position and Flow target. As can be seen in the example figure, a common type definition allows for multiple identical *Object* instances to be created.

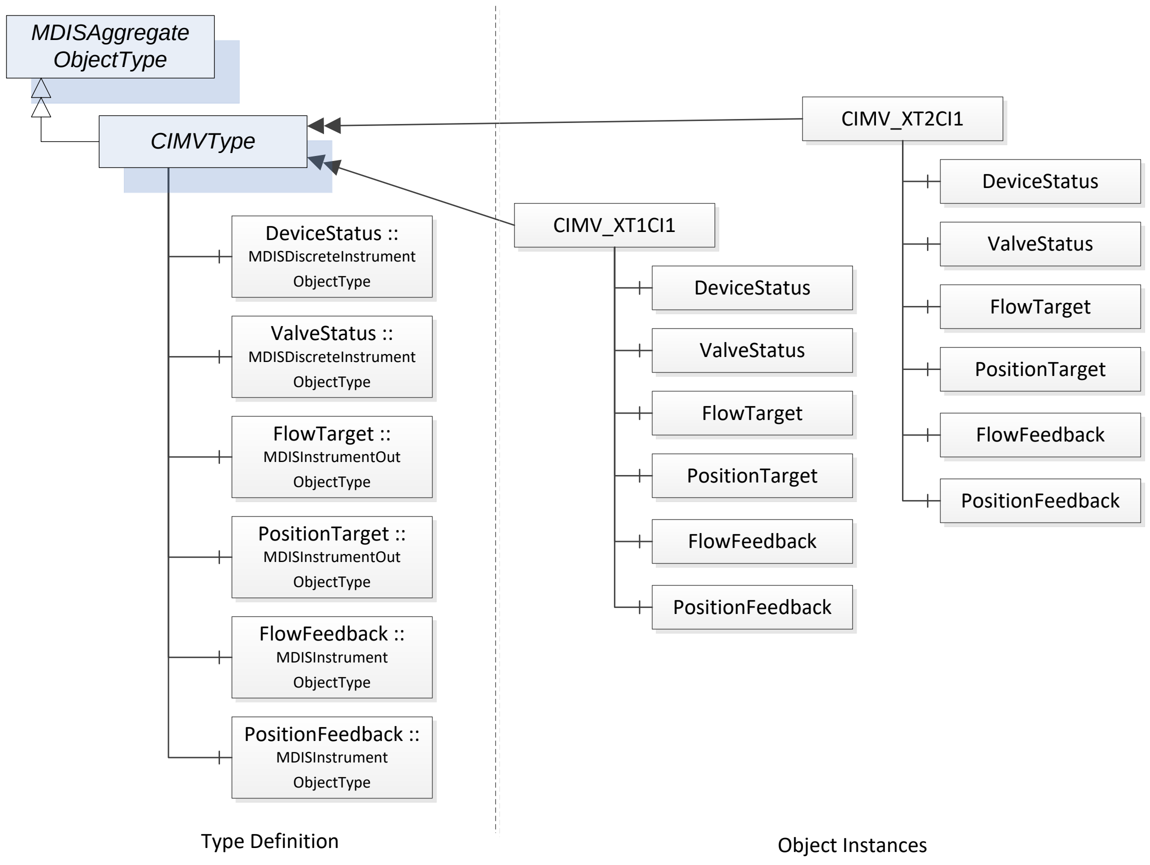
MDISAggregate ObjectType
CIMVType
DeviceStatus :: MDISDiscreteInstrument ObjectType
ValveStatus :: MDISDiscreteInstrument ObjectType
FlowTarget :: MDISInstrumentOut ObjectType
PositionTarget :: MDISInstrumentOut ObjectType
FlowFeedback :: MDISInstrument ObjectType
PositionFeedback :: MDISInstrument ObjectType
CIMV_XT1CI1
CIMV_XT2CI1
DeviceStatus
ValveStatus
FlowTarget
PositionTarget
FlowFeedback
PositionFeedback
Type Definition
Object Instances

Figure 21 - Aggregated Object Type Definition

A vendor does not have to generate aggregate *Objects*; it can just provide a list of the base MDIS *Objects* that are being exposed. For example, a Well might not be configured as a subtype of *MDISAggregateObjectType* it might be a folder that contains the MDIS *Objects* that comprise the Well. If a structure is to be repeated, generating a subtype of the *MDISAggregateObjectType* for the structure can simplify testing and *Client* configuration work.

10 Time Synchronisation

MDIS expects system clocks to be time synchronised. NTP is the recommended method for time synchronisation; however use of alternate time synchronisation method is acceptable. For a system where NTP or other similar time synchronization is not available, the time on the Server can be set using the TimeSyncObject defined in 5.1.11.

11 Redundancy

11.1 General

This section details requirements to standardise redundancy across the MDIS interface. SPCS and DCS should be connected using two segregated communication channels. Designation of the primary communication channel between the SPCS and DCS shall be controlled at the DCS. Automatic reconnection / recovery in the event of a failed communication channel shall be implemented. Failure of either channel shall generate an alarm at the DCS. Communication configurations other than what is defined here may be used. However, it should be specified elsewhere and should be agreed between all parties prior to implementation. Additional redundancy options including physical and logical configuration should be based on required project availability.

In general, redundant communication between the SPCS and DCS shall:

1. Facilitate active redundancy which enables a seamless transfer to a secondary controller in the event of a primary controller failure.

2. Prevent or minimise the loss of subsea production due to a single-component failure or common-mode failure.

The information model available in MDIS OPC UA servers is affected by data arbitration choices (see 4.3.2).

OPC Redundancy is described in the following section.

11.2 OPC UA Redundancy Overview

OPC UA redundancy provides a standard manner for OPC UA *Clients* to determine which OPC UA *Servers* are redundant and the status of the *Servers*. In OPC UA redundant system all redundant servers provide:

- an identical information model,
- an identical instance address space.

The structures, redundancy objects and redundancy behaviours are described in Part 4 – Services and further defined in Part 5 – Information Model.

11.3 OPC UA MDIS Redundancy

In an MDIS system, the information in an OPC UA server maybe identical in some cases. The information model between redundant servers is always identical. MDIS systems typically include arbitration which is described in Section 4.3.2. The arbitration process can result in Servers in which the Servers have an identical instance address space, but it can also result in Servers that do not have identical instance address space.

11.4 MDIS Minimum Requirements

In all MDIS redundant Servers, the standard Redundancy Objects as described in Part 4 and Part 5 shall be supported. In addition, depending on if the data is identical on the servers the following apply:

On *Servers* where all data is identical: The server pair shall support either TRANSPARENT_4 redundancy or HOT_3 redundancy or HOT_AND_MIRRORED_5 redundancy including all associated objects and behaviours as described in Part 4 and Part 5.

- On Servers where not all data is identical: The Server redundancy type cannot be specified and redundancy support shall always identify as NONE_0.

In both cases the *ServerArray* in each *Server* in the redundant set shall list the URL’s for all other *Servers* in the redundant set.

12 OPC UA MDIS Profiles and Conformance Units

12.1 Test requirements

All mandatory *Profiles* shall be certified and any optional *Profiles* that are supported should be certified by an OPC Foundation certification authority (see 1.2.4 for additional details)

12.2 ConformanceUnits

12.2.1 Overview

This section defines *ConformanceUnits* that are specific to the MDIS information model. These *ConformanceUnits* are separated into *ConformanceUnits* that are *Server* specific and those that are *Client* specific.

12.2.2 Server

Table 55 defines the *Server* based *ConformanceUnits*. These units are related to MDIS information models. Table 56 describes general functionality based *ConformanceUnits*.

Table 55 - MDIS Server Information Model ConformanceUnits

Category	Title	Description
Server	MDIS Base Fault	Support the *Fault* flag
Server	MDIS Base FaultCode	Support *FaultCodes*
Server	MDIS Base Warning	Support the *Warning* flag
Server	MDIS Base WarningCode	Support *WarningCodes*
Server	MDIS Base Enabled	Support the *Enabled* flag and the *EnableDisable Method* to toggle the flag.
Server	MDIS Base TagId	Support the *TagId Property*.
Server	MDIS Base Functionality	Supports all required *Namespaces,* queue sizes greater than 1, notification of queue overflows, NodeId and Namespace restrictions
Valve		
Server	MDIS Valve Base	Supports the base required aspect of the *MDISValveObjectType*. This includes position information and the *Move Method* for basic functionality. The Move *Method* basic functionality includes *Direction*, *OverrideInterlocks*, *SEM* and *ShutdownRequest*.
Server	MDIS Valve SignatureRequestStatus	Supports providing information about an existing signature/profile requests, including the request of a profile/signature via the *Move* command.
Server	MDIS Valve CommandRejected	Supports the *CommandRejected*
Server	MDIS Valve LastCommand	Supports the *LastCommand*.
Server	MDIS Valve DefeatableCloseInterlock	Supports information related to *DefeatableCloseInterlock*. This includes *DefeatableCloseInterlock* flag and providing at least one *InterlockFor* reference to a *Variable* of *InterlockVariableType*.
Server	MDIS Valve DefeatableOpenInterlock	Supports information related to *DefeatableOpenInterlock*. This includes *DefeatableOpenInterlock* flag and providing at least one *InterlockFor* reference to a *Variable* of *InterlockVariableType*.
Server	MDIS Valve NonDefeatableCloseInterlock	Supports information related to *NonDefeatableCloseInterlock*. This includes *NonDefeatableCloseInterlock* flag and providing at least one *InterlockFor* reference to a *Variable* of *InterlockVariableType*.
Server	MDIS Valve NonDefeatableOpenInterlock	Supports information related to NonDefeatableOpenInterlock. This includes NonDefeatableCloseInterlock flag and providing at least one InterlockFor reference to a variable of *InterlockVariableType*.
Server	MDIS Valve Duration	Supports the inclusion of *OpenTimeDuration* and *CloseTimeDuration* duration information for the valve.
Instrument		
Server	MDIS Instrument Base	Supports the base required aspect of the *MDISInstrumentObjectType*. This includes the *ProcessVariable*. The *ProcessVariable* includes *EURange* and *EngineeringUnits*

Category	**Title**	**Description**
Server	MDIS Instrument Limits	Supports at least one of the following limit flags: *HHlimit*, *Hlimit*, *Llimit*, *LLlimit*. The actual list of supported limits is reported as part of the *ConformanceUnit*.
Server	MDIS Instrument Setpoints	Supports at least one of the following set points: *HHSetPoint*, *HSetPoint*, *LSetPoint*, *LLSetPoint*. The actual list of supported setpoints is reported as part of the *ConformanceUnit*.
Instrument Out		
Server	MDIS Instrument Out Base	Supports the base required aspect of the *MDISInstrumentOutObjectType*. This includes the *ProcessVariable* of the *Object* and the *WriteValue Method*.
Discrete Instrument		
Server	MDIS Discrete Instrument Base	Supports the base required aspect of the *MDISDiscreteInstrumentObjectType*. This includes the *State* of the *Object*.
Discrete Instrument Out		
Server	MDIS Discrete Out Base	Supports the base required aspect of the *MDISDiscreteOutObjectType*. This includes the *State* of the *Object* and the *WriteValue Method*.
Digital Instrument		
Server	MDIS Digital Instrument Base	Supports the base required aspect of the *MDISDigitalInstrumentObjectType*. This includes the *State* of the *Object*.
Digital Instrument Out		
Server	MDIS Digital Out Base	Supports the base required aspect of the *MDISDigitalOutObjectType*. This includes the *State* of the *Object* and the *WriteState Method*.
Choke		
Server	MDIS Choke Base	Supports the base required aspect of the *MDISChokeObjectType*. This includes *CalculatedPosition* information, *Moving* flag, the *Move Method* (*ChokeMoveMethod*), *Abort Method* (*ChokeAbortMethod*), The *Move Method* basic functionality includes *Position*, *OverrideInterlocks*, and *SEM*. The *SetCalculatedPosition Method* basic functionality includes *CalculatedPosition* and if required the SetCalculatedPositionStatus
Server	MDIS choke SetCalculatedPositionStatus	Supports the optional variable SetCalculatedPositionStatus
Server	MDIS Choke CommnadRejected	Supports the *CommandRejected.*
Server	MDIS Choke DefeatableCloseInterlock	Supports the optional aspect of the *MDISChokeObjectType* related to *DefeatableCloseInterlock*. This includes *DefeatableCloseInterlock* flag and providing at least one *InterlockFor* reference to a *Variable* of *InterlockVariableType*.
Server	MDIS Choke DefeatableOpenInterlock	Supports the optional aspect of the *MDISChokeObjectType* related to *DefeatableOpenInterlock*. This includes *DefeatableOpenInterlock* flag and providing at least one *InterlockFor* reference to a *Variable* of *InterlockVariableType*.
Server	MDIS Choke NonDefeatableCloseInterlock	Supports the optional aspect of the *MDISChokeObjectType* related to *NonDefeatableCloseInterlock*. This includes *NonDefeatableCloseInterlock* flag and providing at least one *InterlockFor* reference to a *Variable* of *InterlockVariableType*.
Server	MDIS Choke NonDefeatableOpenInterlock	Supports the optional aspect of the *MDISChokeObjectType* related to *NonDefeatableOpenInterlock*. This includes *NonDefeatableOpenInterlock* flag and providing at least one *InterlockFor* reference to a *Variable* of *InterlockVariableType*.
Server	MDIS Choke Step Duration	Supports the inclusion of *StepDurationOpen* and *StepDurationClose* time information for the *MDISChokeObjectType*.
Server	MDIS Choke Total Steps	Supports the inclusion of the TotalSteps for the *MDISChokeObjectType*.
Server	MDIS Choke Step Method	Supports the inclusion of the Step (ChokeStepMethod) and the PositionInSteps for the *MDISChokeObjectType*. The Step *Method* basic functionality includes Direction, Steps, OverrideInterlocks, and SEM.
TimeSync Object		
Server	MDIS Timesync Object	Supports the *TimeSync Object*, including allow a client to set the server time.
Information		

Category	Title	Description
Server	MDIS Information version	Supports version information for the MDIS *Server*
Server	MDIS Information Signatures	Supports providing Signatures as *FileType Objects*

Table 56 – MDIS Server Behaviour ConformanceUnits

Category	Title	Description
Server	MDIS Redundancy Base	Supports one of the optional redundancy behaviours
Server	MDIS Redundancy None	Supports MDIS defined side by side redundancy. *Server* can provide side by side redundant data as specified including identical *NodeIds* or naming conventions as required.
Server	MDIS Redundancy Hot	Supports the OPC UA defined redundancy concept of "Hot"
Server	MDIS Redundancy HotPlusMirrored	Supports the OPC UA defined redundancy concept of "HotPlusMirrored"
Server	MDIS Redundancy Transparent	Supports the OPC UA defined redundancy concept of "Transparent"

Table 57 – MDIS Server Aggregate & Extension ConformanceUnits

Category	Title	Description
Server	MDIS Aggregate Object	The MDIS *Server* supports aggregate *Objects* based on *MDISAggregateObjectType*.
Server	MDIS Extension	The *Server* defines new *ObjectTypes* that are extensions of the existing MDIS *ObjectTypes* (*MDISValveObjectType*, *MDISChokeObjectType*, *MDISDigitalInstrumentObjectType*, *MDISDiscreteInstrumentObjectType*, *MDISInstrumentObjectType* or any subtype of these types).

12.2.3 Client

Table 58 defines the Client based *ConformanceUnits*. These units are related to MDIS *InformationModels*.

Table 59 describes general functionality based *ConformanceUnits*

Table 58 - MDIS Client Information Model ConformanceUnits

Category	**Title**	**Description**
Client	MDIS Client Base Fault	The *Client* actively monitors the value of the *Fault* flag and reports the value.
Client	MDIS Client Base FaultCode	The *Client* displays or reports *FaultCode*.
Client	MDIS Client Base Warning	The *Client* makes use of and displays or reports the *Warning* flag.
Client	MDIS Client Base WarningCode	The *Client* displays or reports *WarningCode*.
Client	MDIS Client Base Enabled	The *Client* makes use of and displays or reports the *Enabled* flag and the *EnableDisable Method*.
Client	MDIS Client Base TagId	The *Client* can display or report the *TagId Property*.
Valve		
Client	MDIS Valve Client Base	Uses the base required aspect of the *MDISValveObjectType*. This includes *position* information and the *Move Method* for basic functionality. The Move *Method* basic functionality includes *Direction*, *OverrideInterlocks*, SEM and *ShutdownRequest*. If present the CommandRejected is handled
Client	MDIS Valve Client SignatureRequestStatu s	Makes use of signature/profile information and asks for signature/profile via the *Move* command.
Client	MDIS Valve Client LastCommand	Makes use of the *LastCommand*.
Client	MDIS Valve Client DefeatableCloseInterlo ck	Makes use of information related to *DefeatableCloseInterlock*. This includes *DefeatableCloseInterlock* flag and examining the instance of *InterlockVariableType* referenced by the *InterlockFor* reference.
Client	MDIS Valve Client DefeatableOpenInterlo ck	Makes use of information related to *DefeatableOpenInterlock*. This includes *DefeatableOpenInterlock* flag and examining the instance of *InterlockVariableType* reference by the *InterlockFor* reference.
Client	MDIS Valve Client NonDefeatableCloseInt erlock	Makes use of information related to *NonDefeatableCloseInterlock*. This includes *NonDefeatableCloseInterlock* flag and examining the instance of *InterlockVariableType* reference by the *InterlockFor* reference.
Client	MDIS Valve Client NonDefeatableOpenInt erlock	Makes use of information related to *NonDefeatableOpenInterlock*. This includes *NonDefeatableOpenInterlock* flag and examining the instance of *InterlockVariableType* reference by the *InterlockFor* reference.
Client	MDIS Valve Client Duration	Makes use of the *OpenTimeDuration* and *CloseTimeDuration* duration information for the valve.
Instrument		
Client	MDIS Instrument Client Base	Makes use of the base required aspects of instance of the *MDISInstrumentObjectType*. This includes the *ProcessVariable*.
Client	MDIS Instrument Client Limits	Makes use of at least one of the following limit flags: *HHlimit*, *Hlimit*, *Llimit*, *LLlimit*.
Client	MDIS Instrument Client Setpoints	Makes use of at least one of the following set points: HHSetPoint, HSetPoint, LSetPoint, LLSetPoint.
Instrument Out		
Client	MDIS Instrument Out Client Base	Makes use of the base required aspect of the *MDISInstrumentOutObjectType*. This includes the *ProcessVariable* of the *Object* and the WriteValue *Method*.
Discrete		
Client	MDIS Discrete Instrument Client Base	Makes use of the base required aspect of the *MDISDiscreteInstrumentObjectType*. This includes the *State* of the *Object*.
Discrete Out		
Client	MDIS Discrete Out Client Base	Makes use of the base required aspect of the *MDISDiscreteOutObjectType*. This includes the *State* of the *Object* and the *WriteValue Method*.
Digital		

Category	**Title**	**Description**
Client	MDIS Digital Instrument Client Base	Makes use of the base required aspect of the *MDISDigitalInstrumentObjectType*. This includes the *State* of the *Object*.
Digital Out		
Client	MDIS Digital Out Client Base	Makes use of the base required aspect of the *MDISDigitalOutObjectType*. This includes the *State* of the *Object* and the *WriteState Method*.
Choke		
Client	MDIS Choke Client Base	Makes use of the base required aspect of the *MDISChokeObjectType*. This includes *CalculatedPosition* information, *Moving* flag, the *Move* (*ChokeMoveMethod*), *Abort* (*ChokeAbortMethod*), SetCalculatedPosition (*ChokeSetCalculatedPositionMethod*). The *Move Method* basic functionality includes *Position*, *OverrideInterlocks*, and SEM, The *SetCalculatedPosition Method* basic functionality includes CalculatedPosition and if required the SetCalculatePositionStatus. If present the CommandRejected is handled.
Client	MDIS Choke Client DefeatableCloseInterlock	Makes use of information related to *DefeatableCloseInterlock*. This includes *DefeatableCloseInterlock* flag and examining the instance of *InterlockVariableType* referenced by the *InterlockFor* reference.
Client	MDIS Choke Client DefeatableOpenInterlock	Makes use of information related to *DefeatableOpenInterlock*. This includes *DefeatableOpenInterlock* flag and examining the instance of *InterlockVariableType* referenced by the *InterlockFor* reference.
Client	MDIS Choke Client NonDefeatableCloseInterlock	Makes use of information related to *NonDefeatableCloseInterlock*. This includes *NonDefeatableCloseInterlock* flag and examining the instance of *InterlockVariableType* referenced by the *InterlockFor* reference.
Client	MDIS Choke Client NonDefeatableOpenInterlock	Makes use of information related to *NonDefeatableOpenInterlock*. This includes *NonDefeatableOpenInterlock* flags and examining the instance of *InterlockVariableType* referenced by the *InterlockFor* reference.
Client	MDIS Choke Client Step duration	Makes use of the *StepDurationOpen* and *StepDurationClose* time information for the *MDISChokeObjectType*.
Client	MDIS Choke Client Total Steps	Makes use of the TotalSteps for the *MDISChokeObjectType*.
Client	MDIS Choke Client Step method	Makes use of the Step (*ChokeStepMethod*) and the PositionInSteps for the Choke. The Step *Method* basic functionality includes Direction, Steps, OverrideInterlocks, and SEM.
TimeSync Object		
Client	MDIS TimeSync Client	Can be configured to call the Timesync Object at a periodic rate, providing time synchronization to the server
Information		
Client	MDIS Signature Client	The Client can access the instance of FileType objects to obtain signatures
Client	MDIS Information Client	The Client makes use of the version information to identify supported functionality of the server, including handling server that are of different versions.

Table 59 - MDIS Client Behaviour ConformanceUnits

Category	Title	Description
Client	MDIS Client Redundancy	Can communicate with a MDIS *Server* that transmits data redundantly. Selecting appropriate channel, handling *Server* failovers and generally supporting all specified actions.

Table 60 - MDIS Client Aggregation & Extension ConformanceUnits

Category	Title	Description
Client	MDIS Client Aggregate	The *Client* can process and / or display information from an instance of an *MDISAggregateObjectType* subtype on a *Server*.
Client	MDIS Client Extension	The *Client* can process and / or display information from an instance of Extension types defined by the *Server*. This includes extension to all of the existing type and subtypes of them.
Client	MDIS Client Extension Extra	The *Client* can process and / or display information from the extended fields in an Extension *Object* defined in a *Server*, without programming changes, i.e. only requiring configuration changes

12.3 Facet

12.3.1 Overview

The section describes the various *Facets* that are provided as part of the MDIS OPC UA *InformationModel*. These *Facets* include MDIS *InformationModel ConformanceUnits*, but they also include *ConformanceUnits* or *Facets* from the Part 7 – Profiles specification. They are summarised in Table 61

Table 61 - MDIS Profiles and Facets

Profile	Related Category	URI
MDIS Base Functionality Server Facet	MDIS	http://opcfoundation.org/UA-Profile/Server/MDIS/BaseFunctionServer
MDIS Valve Model Server Facet	MDIS	http://opcfoundation.org/UA-Profile/Server/MDIS/ValveModel
MDIS Instrument Model Server Facet	MDIS	http://opcfoundation.org/UA-Profile/Server/MDIS/InstrumentModel
MDIS Instrument Out Model Server Facet	MDIS	http://opcfoundation.org/UA-Profile/Server/MDIS/InstrumentOutModel
MDIS Choke Model Server Facet	MDIS	http://opcfoundation.org/UA-Profile/Server/MDIS/ChokeModel
MDIS Discrete Model Server Facet	MDIS	http://opcfoundation.org/UA-Profile/Server/MDIS/DiscreteModel
MDIS Discrete Out Model Server Facet	MDIS	http://opcfoundation.org/UA-Profile/Server/MDIS/DiscreteOutModel
MDIS Digital Model Server Facet	MDIS	http://opcfoundation.org/UA-Profile/Server/MDIS/DigitalModel
MDIS Digital Out Model Server Facet	MDIS	http://opcfoundation.org/UA-Profile/Server/MDIS/DigitalOutModel
MDIS Redundancy Server Facet	MDIS	http://opcfoundation.org/UA-Profile/Server/MDIS/Redundancy
MDIS Aggregate Object Server Facet	MDIS	http://opcfoundation.org/UA-Profile/Server/MDIS/AggregateObject
MDIS Extension Object Server Facet	MDIS	http://opcfoundation.org/UA-Profile/Server/MDIS/ExtensionObject
MDIS Signature Transfer Server Facet	MDIS	http://opcfoundation.org/UA-Profile/Server/MDIS/SignatureTransfer
MDIS TimeSync Object Server Facet	MDIS	http://opcfoundation.org/UA-Profile/Server/MDIS/TimeSyncObject
MDIS Base Client Facet	MDIS	http://opcfoundation.org/UA-Profile/Client/MDIS/Base
MDIS Valve Model Client Facet	MDIS	http://opcfoundation.org/UA-Profile/Client/MDIS/Valve
MDIS Instrument Model Client Facet	MDIS	http://opcfoundation.org/UA-Profile/Client/MDIS/Instrument
MDIS Instrument Out Model Client Facet	MDIS	http://opcfoundation.org/UA-Profile/Client/MDIS/InstrumentOut
MDIS Choke Model Client Facet	MDIS	http://opcfoundation.org/UA-Profile/Client/MDIS/Choke
MDIS Discrete Model Client Facet	MDIS	http://opcfoundation.org/UA-Profile/Client/MDIS/Discrete
MDIS Discrete Out Model Client Facet	MDIS	http://opcfoundation.org/UA-Profile/Client/MDIS/DiscreteOut

Profile	**Related Category**	**URI**
MDIS Digital Model Client Facet	MDIS	http://opcfoundation.org/UA-Profile/Client/MDIS/Digital
MDIS Digital Out Model Client Facet	MDIS	http://opcfoundation.org/UA-Profile/Client/MDIS/DigitalOut
MDIS Redundancy Client Facet	MDIS	http://opcfoundation.org/UA-Profile/Client/MDIS/Redundancy
MDIS Aggregate Object Client Facet	MDIS	http://opcfoundation.org/UA-Profile/Client/MDIS/AggregateObject
MDIS Extension Object Client Facet	MDIS	http://opcfoundation.org/UA-Profile/Client/MDIS/ExtensionObject
MDIS Extension Extra Object Client Facet	MDIS	http://opcfoundation.org/UA-Profile/Client/MDIS/ExtensionExtra
MDIS Signature Transfer Client Facet	MDIS	http://opcfoundation.org/UA-Profile/Client/MDIS/SignatureTransfer
MDIS TimeSync Object Client Facet	MDIS	http://opcfoundation.org/UA-Profile/Client/MDIS/TimeSyncObject
MDIS Solution Client Profile	MDIS	http://opcfoundation.org/UA-Profile/Client/MDIS/Solution
MDIS Solution Server Profile	MDIS	http://opcfoundation.org/UA-Profile/Server/MDIS/Solution

12.3.2 Server

12.3.2.1 MDIS Base Functionality Server Facet

Table 62 defines a *Facet* that describes the base characteristics that all OPC UA Servers shall support, if they support the MDIS companion specification.

Table 62 – MDIS Base Functionality Server Facet

Group	**Conformance Unit / Profile Title**	**Optional**
Profile	Standard DataChange Subscription Server Facet	
Profile	Core Server Facet	
Profile	UA-TCP UA-SC UA Binary	
Profile	Data Access Server Facet	
Monitored Item Services	Monitor MinQueueSize_05	False
Profile	Method Server Facet	
Profile	Security Time Synchronization	
MDIS Model	MDIS Base Functionality	False
MDIS Model	MDIS Information Version	True

This *Profile* includes a number of *Profiles* and *ConformanceUnits*.

12.3.2.2 MDIS Valve Model Server Facet

Table 63 defines a *Facet* that describes the base characteristics for an OPC UA *Server* that is exposing the *MDISValveObjectType* model.

Table 63 - MDIS Valve Model Server Facet

Group	**Conformance Unit / Profile Title**	**Optional**
MDIS Model	MDIS Base Fault	False
MDIS Model	MDIS Base FaultCode	True
MDIS Model	MDIS Base Warning	True
MDIS Model	MDIS Base WarningCode	True
MDIS Model	MDIS Base Enabled	True
MDIS Model	MDIS Base TagId	True
MDIS Model	MDIS Valve Base	False

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Valve CommandRejected	True
MDIS Model	MDIS Valve SignatureRequest	True
MDIS Model	MDIS Valve LastCommand	True
MDIS Model	MDIS Valve DefeatableCloseInterlock	True
MDIS Model	MDIS Valve DefeatableOpenInterlock	True
MDIS Model	MDIS Valve NonDefeatableCloseInterlock	True
MDIS Model	MDIS Valve NonDefeatableOpenInterlock	True
MDIS Model	MDIS Valve Duration	True
MDIS Model	MDIS Information signatures	True

12.3.2.3 MDIS Instrument Model Server Facet

Table 64 defines a *Facet* that describes the base characteristics for an OPC UA *Server* that is exposing the *MDISInstrumentObjectType* model.

Table 64 - MDIS Instrument Model Server Facet

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Base Fault	False
MDIS Model	MDIS Base FaultCode	True
MDIS Model	MDIS Base Warning	True
MDIS Model	MDIS Base WarningCode	True
MDIS Model	MDIS Base Enabled	True
MDIS Model	MDIS Base TagId	True
MDIS Model	MDIS Instrument Base	False
MDIS Model	MDIS Instrument Limits	True
MDIS Model	MDIS Instrument Setpoints	True

12.3.2.4 MDIS Instrument Out Model Server Facet

Table 65 defines a *Facet* that describes the base characteristics for an OPC UA *Server* that is exposing the *MDISInstrumentOutObjectType* model.

Table 65 - MDIS Instrument Out Model Server Facet

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Base Fault	False
MDIS Model	MDIS Base FaultCode	True
MDIS Model	MDIS Base Warning	True
MDIS Model	MDIS Base WarningCode	True
MDIS Model	MDIS Base Enabled	True
MDIS Model	MDIS Base TagId	True
MDIS Model	MDIS Instrument Out Base	False
MDIS Model	MDIS Instrument Limits	True
MDIS Model	MDIS Instrument Set points	True

12.3.2.5 MDIS Discrete Model Server Facet

Table 66 defines a *Facet* that describes the base characteristics for an OPC UA *Server* that is exposing the *MDISDiscreteInstrumentObjectType* model.

Table 66 - MDIS Discrete Model Server Facet

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Base Fault	False
MDIS Model	MDIS Base FaultCode	True
MDIS Model	MDIS Base Warning	True
MDIS Model	MDIS Base WarningCode	True
MDIS Model	MDIS Base Enabled	True
MDIS Model	MDIS Base TagId	True
MDIS Model	MDIS Discrete Instrument Base	False

12.3.2.6 MDIS Discrete Out Model Server Facet

Table 67 defines a *Facet* that describes the base characteristics for an OPC UA *Server* that is exposing the *MDISDiscreteOutObjectType* model.

Table 67 - MDIS Discrete Out Model Server Facet

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Base Fault	False
MDIS Model	MDIS Base FaultCode	True
MDIS Model	MDIS Base Warning	True
MDIS Model	MDIS Base WarningCode	True
MDIS Model	MDIS Base Enabled	True
MDIS Model	MDIS Base TagId	True
MDIS Model	MDIS Discrete Out Base	False

12.3.2.7 MDIS Digital Model Server Facet

Table 68 defines a *Facet* that describes the base characteristics for an OPC UA *Server* that is exposing the *MDISDigitalInstrumentObjectType*.

Table 68 - MDIS Digital Model Server Facet

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Base Fault	False
MDIS Model	MDIS Base FaultCode	True
MDIS Model	MDIS Base Warning	True
MDIS Model	MDIS Base WarningCode	True
MDIS Model	MDIS Base Enabled	True
MDIS Model	MDIS Base TagId	True
MDIS Model	MDIS Digital Instrument Base	False

12.3.2.8 MDIS Digital Out Model Server Facet

Table 69 defines a *Facet* that describes the base characteristics for an OPC UA *Server* that is exposing the *MDISDigitalOutObjectType*.

Table 69 - MDIS Digital Out Model Server Facet

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Base Fault	False
MDIS Model	MDIS Base FaultCode	True

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Base Warning	True
MDIS Model	MDIS Base WarningCode	True
MDIS Model	MDIS Base Enabled	True
MDIS Model	MDIS Base TagId	True
MDIS Model	MDIS Digital Out Base	False

12.3.2.9 MDIS Choke Model Server Facet

Table 70 defines a *Facet* that describes the base characteristics for an OPC UA *Server* that is exposing the *MDISChokeObjectType*.

Table 70 - MDIS Choke Model Server Facet

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Base Fault	False
MDIS Model	MDIS Base FaultCode	True
MDIS Model	MDIS Base Warning	True
MDIS Model	MDIS Base WarningCode	True
MDIS Model	MDIS Base Enabled	True
MDIS Model	MDIS Base TagId	True
MDIS Model	MDIS Choke Base	False
MDIS Model	MDIS Choke DefeatableCloseInterlock	True
MDIS Model	MDIS Choke DefeatableOpenInterlock	True
MDIS Model	MDIS Choke NonDefeatableCloseInterlock	True
MDIS Model	MDIS Choke NonDefeatableOpenInterlock	True
MDIS Model	MDIS Choke Step Duration	True
MDIS Model	MDIS Choke Total Steps	True
MDIS Model	MDIS Choke Step method	True
MDIS Model	MDIS Choke CommandRejected	True
MDIS Model	MDIS Choke SetCalculatedPositionStatus	True

12.3.2.10 MDIS Redundancy Server Facet

Table 71 defines a *Facet* that describes Redundancy functionality that a Server would support. The *Server* must support at least one of the optional conformance units

Table 71 - MDIS Redundancy Server Facet

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Redundancy Base	False
MDIS Model	MDIS Redundancy None	True
MDIS Model	MDIS Redundancy Hot	True
MDIS Model	MDIS Redundancy HotPlusMirrored	True
MDIS Model	MDIS Redundancy Transparent	True

12.3.2.11 MDIS Aggregate Object Server Facet

Table 72 defines a *Facet* that describes Aggregate functionality based on *MDISAggregateObjectType* that a Server would support.

Table 72 - MDIS Aggregate Object Server Facet

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Aggregate Object	False

12.3.2.12 MDIS Extension Object Server Facet

Table 73 defines a *Facet* that describes Object extension functionality that a *Server* would support.

Table 73 - MDIS Extension Object Server Facet

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Extension	False

12.3.2.13 MDIS Signature Transfer Server Facet

Table 74 defines a *Facet* that describes Signature transfer functionality that a *Server* would support.

Table 74 - MDIS Signature Transfer Server Facet

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Information Signatures	False

12.3.2.14 MDIS TimeSync Object Server Facet

Table 75 defines a *Facet* that describes TimeSync Object extension functionality that a *Server* would support.

Table 75 - MDIS TimeSync Object Server Facet

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Timesync Object	False

12.3.3 Client

12.3.3.1 MDIS Base Client Facet

Table 76 defines a *Facet* that describes the base characteristics for all OPC UA *Clients* that make use of this companion specification. Additional *Profiles* will define support for various object models that are part of this specification.

Table 76 - MDIS Base Client Facet

Group	Conformance Unit / Profile Title	Optional
Profile	AddressSpace Lookup Client Facet	
Profile	DataAccess Client Facet	
Profile	DataChange Subscriber Client Facet	
Profile	Method Client Facet	
Profile	UA-TCP UA-SC UA Binary	
Profile	Security Time Synchronisation	
Session Services	Session Client Base	False
Session Services	Session Client Renew NodeIds	False
Session Services	Session Client KeepAlive	False
Session Services	Session Client Detect Shutdown	False
MDIS Model	MDIS Information Client	

12.3.3.2 MDIS Valve Model Client Facet

Table 77 defines a *Facet* that describes the base characteristics for an OPC UA *Client* using the *MDISValveObjectType* model.

Table 77 - MDIS Valve Model Client Facet

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Client Base Fault	False
MDIS Model	MDIS Client Base FaultCode	True
MDIS Model	MDIS Client Base Warning	True
MDIS Model	MDIS Client Base WarningCode	True
MDIS Model	MDIS Client Base Enabled	True
MDIS Model	MDIS Client Base TagId	True
MDIS Model	MDIS Valve Client Base	False
MDIS Model	MDIS Valve Client SignatureRequestStatus	True
MDIS Model	MDIS Valve Client LastCommand	True
MDIS Model	MDIS Valve Client DefeatableCloseInterlock	True
MDIS Model	MDIS Valve Client DefeatableOpenInterlock	True
MDIS Model	MDIS Valve Client NonDefeatableCloseInterlock	True
MDIS Model	MDIS Valve Client NonDefeatableOpenInterlock	True
MDIS Model	MDIS Valve Client Duration	True

12.3.3.3 MDIS Instrument Model Client Facet

Table 78 defines a *Facet* that describes the base characteristics for an OPC UA *Client* using the *MDISInstrumentObjectType* model.

Table 78 - MDIS Instrument Model Client Facet

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Client Base Fault	False
MDIS Model	MDIS Client Base FaultCode	True
MDIS Model	MDIS Client Base Warning	True
MDIS Model	MDIS Client Base WarningCode	True
MDIS Model	MDIS Client Base Enabled	True
MDIS Model	MDIS Client Base TagId	True
MDIS Model	MDIS Instrument Client Base	False
MDIS Model	MDIS Instrument Client Limits	True
MDIS Model	MDIS Instrument Client Setpoints	True

12.3.3.4 MDIS Instrument Out Model Client Facet

Table 79 defines a *Facet* that describes the base characteristics for an OPC UA *Client* using the *MDISInstrumentOutObjectType* model.

Table 79 - MDIS Instrument Out Model Client Facet

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Client Base Fault	False
MDIS Model	MDIS Client Base FaultCode	True
MDIS Model	MDIS Client Base Warning	True

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Client Base WarningCode	True
MDIS Model	MDIS Client Base Enabled	True
MDIS Model	MDIS Client Base TagId	True
MDIS Model	MDIS Instrument Client Base	False
MDIS Model	MDIS Instrument Client Limits	True
MDIS Model	MDIS Instrument Client Setpoints	True
MDIS Model	MDIS Instrument Out Client Base	False

12.3.3.5 MDIS Discrete Model Client Facet

Table 80 defines a *Facet* that describes the base characteristics for an OPC UA *Client* using the *MDISDiscreteInstrumentObjectType* model.

Table 80 - MDIS Discrete Model Client Facet

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Client Base Fault	False
MDIS Model	MDIS Client Base FaultCode	True
MDIS Model	MDIS Client Base Warning	True
MDIS Model	MDIS Client Base WarningCode	True
MDIS Model	MDIS Client Base Enabled	True
MDIS Model	MDIS Client Base TagId	True
MDIS Model	MDIS Discrete Client Base	False

12.3.3.6 MDIS Discrete Out Model Client Facet

Table 81 defines a *Facet* that describes the base characteristics for an OPC UA *Client* using the *MDISDiscreteOutObjectType*.

Table 81 - MDIS Discrete Out Model Client Facet

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Client Base Fault	False
MDIS Model	MDIS Client Base FaultCode	True
MDIS Model	MDIS Client Base Warning	True
MDIS Model	MDIS Client Base WarningCode	True
MDIS Model	MDIS Client Base Enabled	True
MDIS Model	MDIS Client Base TagId	True
MDIS Model	MDIS Discrete Client Base	False
MDIS Model	MDIS Discrete Out Client Base	False

12.3.3.7 MDIS Digital Model Client Facet

Table 82 defines a *Facet* that describes the base characteristics for an OPC UA *Client* using the *MDISDigitalInstrumentObjectType* model.

Table 82 - MDIS Digital Model Client Facet

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Client Base Fault	False
MDIS Model	MDIS Client Base FaultCode	True

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Client Base Warning	True
MDIS Model	MDIS Client Base WarningCode	True
MDIS Model	MDIS Client Base Enabled	True
MDIS Model	MDIS Client Base TagId	True
MDIS Model	MDIS Digital Client Base	False

12.3.3.8 MDIS Digital Out Model Client Facet

Table 83 defines a *Facet* that describes the base characteristics for an OPC UA *Client* using the *MDISDigitalOutObjectType* model.

Table 83 - MDIS Digital Out Model Client Facet

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Client Base Fault	False
MDIS Model	MDIS Client Base FaultCode	True
MDIS Model	MDIS Client Base Warning	True
MDIS Model	MDIS Client Base WarningCode	True
MDIS Model	MDIS Client Base Enabled	True
MDIS Model	MDIS Client Base TagId	True
MDIS Model	MDIS Digital Client Base	False
MDIS Model	MDIS Digital Out Client Base	False

12.3.3.9 MDIS Choke Model Client Facet

Table 84 defines a *Facet* that describes the base characteristics for an OPC UA *Client* using the *MDISChokeObjectType* model.

Table 84 - MDIS Choke Model Client Facet

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Client Base Fault	False
MDIS Model	MDIS Client Base FaultCode	True
MDIS Model	MDIS Client Base Warning	True
MDIS Model	MDIS Client Base WarningCode	True
MDIS Model	MDIS Client Base Enabled	True
MDIS Model	MDIS Client Base TagId	True
MDIS Model	MDIS Choke Client Base	False
MDIS Model	MDIS Choke Client DefeatableCloseInterlock	True
MDIS Model	MDIS Choke Client DefeatableOpenInterlock	True
MDIS Model	MDIS Choke Client NonDefeatableCloseInterlock	True
MDIS Model	MDIS Choke Client NonDefeatableOpenInterlock	True
MDIS Model	MDIS Choke Client Step duration	True
MDIS Model	MDIS Choke Client Total Steps	True
MDIS Model	MDIS Choke Client Step method	True

12.3.3.10 MDIS Redundancy Client Facet

Table 85 defines a *Facet* that describes Redundancy that a *Client* would support.

Table 85 - MDIS Redundancy Client Facet

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Client Redundancy	False

12.3.3.11 MDIS Aggregate Object Client Facet

Table 86 defines a *Facet* that describes Aggregate Object functionality that a *Client* would support.

Table 86 - MDIS Aggregate Object Client Facet

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Client Aggregate	False

12.3.3.12 MDIS Extension Object Client Facet

Table 87 defines a *Facet* that describes Extension Object functionality that a *Client* would support.

Table 87 - MDIS Extension Object Client Facet

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Client Extension	False

12.3.3.13 MDIS Extension Extra Object Client Facet

Table 88 defines a *Facet* that describes Extension Object extra Fields functionality that a *Client* would support.

Table 88 - MDIS Extension Extra Object Client Facet

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Client Extension Extra	False

12.3.3.14 MDIS Signature Transfer Client Facet

Table 89 defines a *Facet* that describes signature transfer functionality that a *Client* would support.

Table 89 - MDIS Signature Transfer Client Facet

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS Signature Client	False

12.3.3.15 MDIS TimeSync Object Client Facet

Table 90 defines a *Facet* that describes TimeSync functionality that a *Client* would support.

Table 90 - MDIS TimeSync Object Client Facet

Group	Conformance Unit / Profile Title	Optional
MDIS Model	MDIS TimeSync Client	False

12.4 MDIS OPC UA Profiles

12.4.1 Overview

This specification has defined a number of individual *Facets* that a *Server* and / or a *Client* are expected to combine and utilise in an application. The following *Profiles* provide a recommended combination of functionality that a *Server* or *Client* should include. These are complete *Profiles* that include all required *Profiles* and *ConformanceUnits* to implement a *Server* or a *Client*.

12.4.2 MDIS Solution Client Profile

Table 91 defines a Full Featured *Profile* that describes the characteristics for an OPC UA *Client*. The OPC UA *Client* may expose additional functionality as separate *Profiles*.

Table 91 - MDIS Solution Client Profile

Group	Conformance Unit / Profile Title	Optional
Profile	MDIS Base Functionality Client Facet	
Profile	MDIS Valve Model Client Facet	
Profile	MDIS Instrument Model Client Facet	
Profile	MDIS Choke Model Client Facet	
Profile	MDIS Discrete Model Client Facet	
Profile	MDIS Digital Model Client Facet	

12.4.3 MDIS Solution Server Profile

Table 92 defines a full featured *Profile* that describes the base characteristics for an OPC UA *Server*. The OPC UA *Server* may expose additional functionality as *Profiles*.

Table 92 - MDIS Solution Server Profile

Group	Conformance Unit / Profile Title	Optional
Profile	MDIS Base Functionality Server Facet	
Profile	MDIS Valve Model Server Facet	
Profile	MDIS Instrument Model Server Facet	
Profile	MDIS Choke Model Server Facet	
Profile	MDIS Discrete Model Server Facet	
Profile	MDIS Digital Model Server Facet	

12.5 Equipment Certification

The MDIS interface shall be certified at an OPC Foundation Certification Test Laboratory on a product basis. Any major release of the product shall be recertified. Recertification can also be required if there are any changes to the standard or test cases defined by MDIS. The certification documentation shall include

- a list of valid MDIS *Profiles* and optional *ConformanceUnits* for which the vendor equipment is certified,
- vendor equipment information used for testing,
- equipment architecture and configuration used for testing during the certification process.

Extensions or aggregate *Objects* added during a project do not need to be recertified.

13 Namespaces

13.1 Status Codes

Table 93 defines the list of standard *Method* call error codes generated by a MDIS Server.

Table 93 - Method error codes

Result Code	**Description**
Good_Completes_Ascynchronously	See OPC UA Part 4 – Services for the description of this result code. (The *Method* id does not refer to a *Method* for the specified Object.)
Bad_MethodInvalid	See OPC UA Part 4 – Services for the description of this result code. (The *Method* id does not refer to a *Method* for the specified Object.)
Bad_NotImplemented	See OPC UA Part 4 – Services for the description of this result code. (Requested operation is not implemented.)
Bad_NodeIdUnknown	See OPC UA Part 4 – Services for the description of this result code. (Used to indicate that the specified Object is not valid)
Bad_ArgumentsMissing	See OPC UA Part 4 – Services for the description of this result code (The Client did not specify all of the input arguments for the *Method*.)
Bad_TooManyArguments	See OPC UA Part 4 – Services for the description of this result code (The Client specified more input arguments than defined for the *Method*.)
Bad_InvalidArgument	See OPC UA Part 4 – Services for the description of this result code. (Used to indicate in the operation level results that one or more of the input arguments are invalid. The inputArgumentResults contain the specific status code for each invalid argument.)
Bad_TypeMismatch	See OPC UA Part 4 – Services for the description of this result code. (Used to indicate that an input argument does not have the correct data type.)
Bad_OutOfRange	See OPC UA Part 4 – Services for the description of this result code. (Used to indicate that an input argument is outside the acceptable range.)
Bad_Timeout	See OPC UA Part 4 – Services for the description of this result code. (The operation timed out. – the Server did not respond to the command)
Bad_InvalidState	See OPC UA Part 4 – Services for the description of this result code. (The operation cannot be completed because the Object is closed, uninitialized or in some other invalid state.)
Bad_AccessDenied	See OPC UA Part 4 – Services for the description of this result code. (The operation cannot be completed because the server does not allow the client to perform the operation)

13.2 Handling of OPC UA Namespaces

Namespaces are used by OPC UA to create unique identifiers across different naming authorities. The *Attributes NodeId* and *BrowseName* are identifiers. A *Node* in the UA *AddressSpace* is unambiguously identified using a *NodeId*. Unlike *NodeIds*, the *BrowseName* cannot be used to unambiguously identify a *Node*. Different *Nodes* may have the same *BrowseName*. They are used to build a browse path between two *Nodes* or to define a standard *Property*.

Servers may often choose to use the same namespace for the *NodeId* and the *BrowseName*. However, if they want to provide a standard *Property*, its *BrowseName* shall have the namespace of the standards body although the namespace of the *NodeId* reflects something else, for example the *EngineeringUnits Property*. All *NodeIds* of *Nodes* not defined in this specification shall not use the standard namespaces.

Table 94 provides a list of mandatory and optional namespaces used in an <title> OPC UA *Server*.

Table 94 – Namespaces used in a <title> Server

NamespaceURI	**Description**	**Use**
http://opcfoundation.org/UA/	Namespace for *NodeIds* and *BrowseNames* defined in the OPC UA specification. This namespace shall have namespace index 0.	Mandatory
Local Server URI	Namespace for nodes defined in the local server. This may include types and instances used in an AutoID Device represented by the Server. This namespace shall have namespace index 1.	Mandatory
http://opcfoundation.org/UA//MDIS	Namespace for *NodeIds* and *BrowseNames* defined in This specification. The namespace index is *Server* specific.	Mandatory
Vendor specific types	A *Server* may provide vendor-specific types like types derived from *ObjectTypes* defined in this specification in a vendor-specific namespace.	Optional

NamespaceURI	Description	Use
Vendor specific instances	A *Server* provides vendor-specific instances of the standard types or vendor-specific instances of vendor-specific types in a vendor-specific namespace. It is recommended to separate vendor specific types and vendor specific instances into two or more namespaces.	Mandatory

Table 95 provides a list of namespaces and their index used for *BrowseNames* in this specification. The default namespace of this specification is not listed since all *BrowseNames* without prefix use this default namespace.

Table 95 – Namespaces used in this specification

NamespaceURI	Namespace Index	Example
http://opcfoundation.org/UA/	0	0:EngineeringUnits

This section defines the numeric identifiers for all of the numeric *NodeIds* defined by the MDIS OPC UA Specification. The identifiers are specified in a CSV file with the following syntax:

<SymbolName>, <Identifier>, <NodeClass>

Where the *SymbolName* is either the *BrowseName* of a *Type Node* or the *BrowsePath* for an *Instance Node* that appears in the specification and the *Identifier* is numeric value for the *NodeId*.

The *BrowsePath* for an instance *Node* is constructed by appending the *BrowseName* of the instance *Node* to *BrowseName* for the containing instance or type. A '_' character is used to separate each *BrowseName* in the path. For example, OPC UA Part 5 defines the *ServerType ObjectType Node* which has the *NamespaceArray Property*. The *SymbolName* for the NamespaceArray *InstanceDeclaration* within the *ServerType* declaration is: *ServerType_NamespaceArray*. OPC UA Part 5 also defines a standard instance of the *ServerType ObjectType* with the *BrowseName* '*Server*'. The *BrowseName* for the *NamespaceArray Property* of the standard *Server Object* is: *Server_NamespaceArray*.

The CSV associated with this version of the standard can be found here:

http://www.opcfoundation.org/UA/schemas/MDIS/1.2/MDIS.csv

The XML *UANodeSet* file that is a definition of the *InformationModel* generated by this specification. The *UANodeSet* description is available from the OPC Foundation web site (http://www.opcfoundation.org/UA/schemas/MDIS/1.2/OPC.MDIS.NodeSet2.xml) as an XML file. It uses the import/export format defined in OPC UA Part 5. This file can be directly used by a *Server* that wishes to expose the *InformationModel* (types) defined in this specification.

Annex A Sequence Diagrams(Informative)

A.1 Introduction

The following section provides sample sequence diagrams for each of the MDIS *ObjectTypes*. These sample sequences are not mandated or the only valid variant.

A.2 MDIS Discrete Instrument Object Sequence Diagrams

A.2.1 Enable Disable

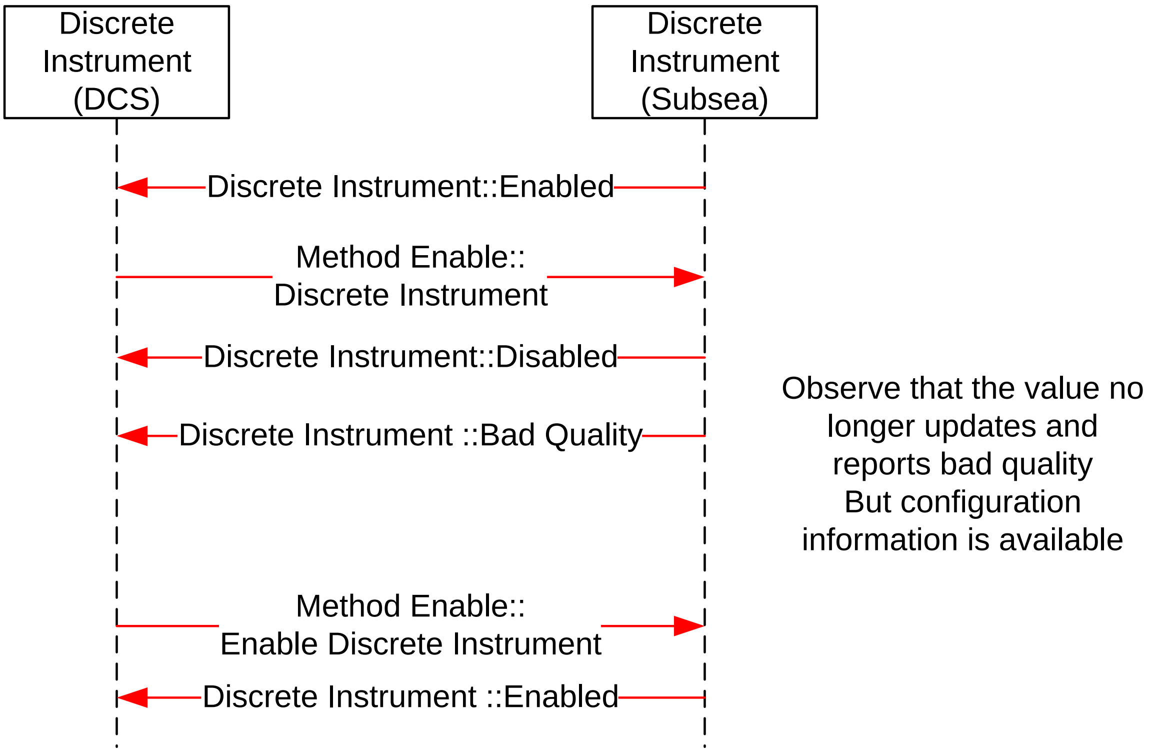
Discrete Instrument (DCS)
Discrete Instrument (Subsea)
Discrete Instrument::Enabled
Method Enable:: Discrete Instrument
Discrete Instrument::Disabled
Discrete Instrument ::Bad Quality
Observe that the value no longer updates and reports bad quality But configuration information is available
Method Enable:: Enable Discrete Instrument
Discrete Instrument ::Enabled

Figure 22 - Discrete Instrument

A.3 MDIS Digital Instrument Object Sequence Diagrams

A.3.1 Enable Disable

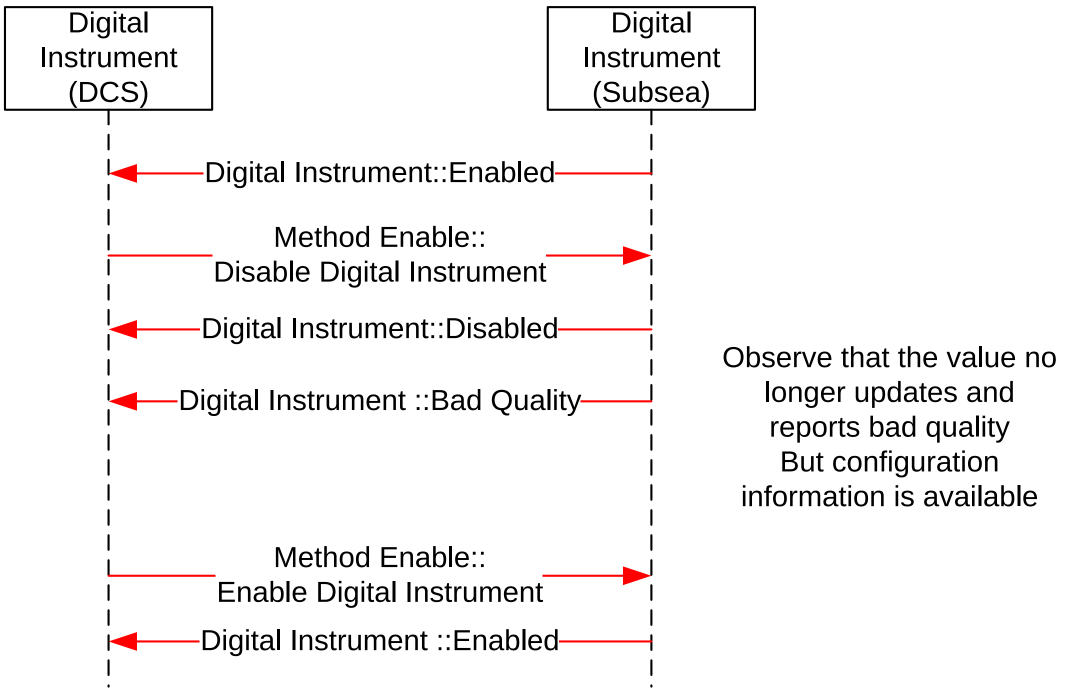
Digital Instrument (DCS)
Digital Instrument (Subsea)
Digital Instrument::Enabled
Method Enable:: Disable Digital Instrument
Digital Instrument::Disabled
Digital Instrument ::Bad Quality
Observe that the value no longer updates and reports bad quality But configuration information is available
Method Enable:: Enable Digital Instrument
Digital Instrument ::Enabled

Figure 23 - Digital Instrument

A.4 MDIS Instrument Object Sequence Diagrams

A.4.1 Enable Disable

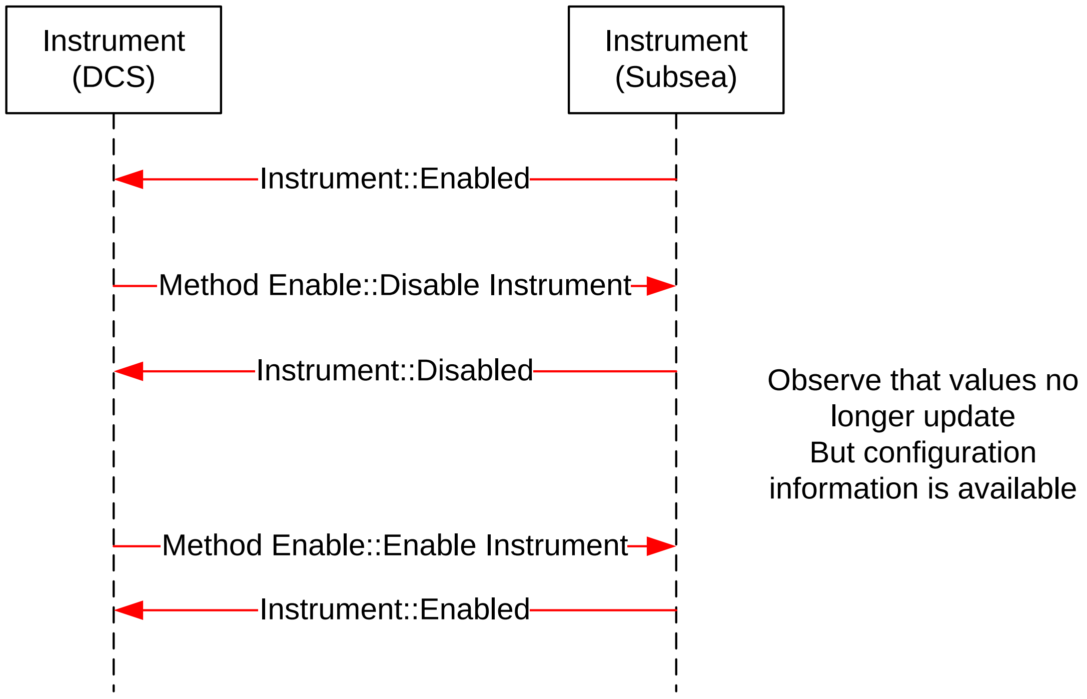
Instrument (DCS)
Instrument (Subsea)
Instrument::Enabled
Method Enable::Disable Instrument
Instrument::Disabled
Observe that values no longer update But configuration information is available
Method Enable::Enable Instrument
Instrument::Enabled

Figure 24 - Instrument Enable / Disable

A.4.2 Write to Setpoint

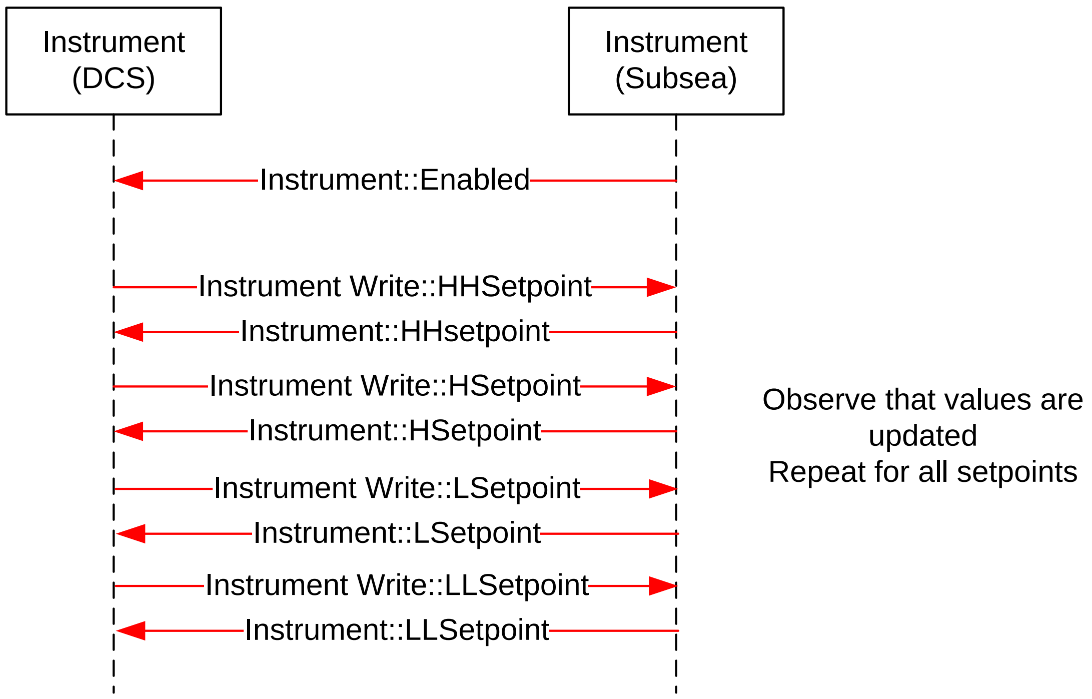

Figure 25 - Instrument Setpoint changes

A.4.3 Display Limits

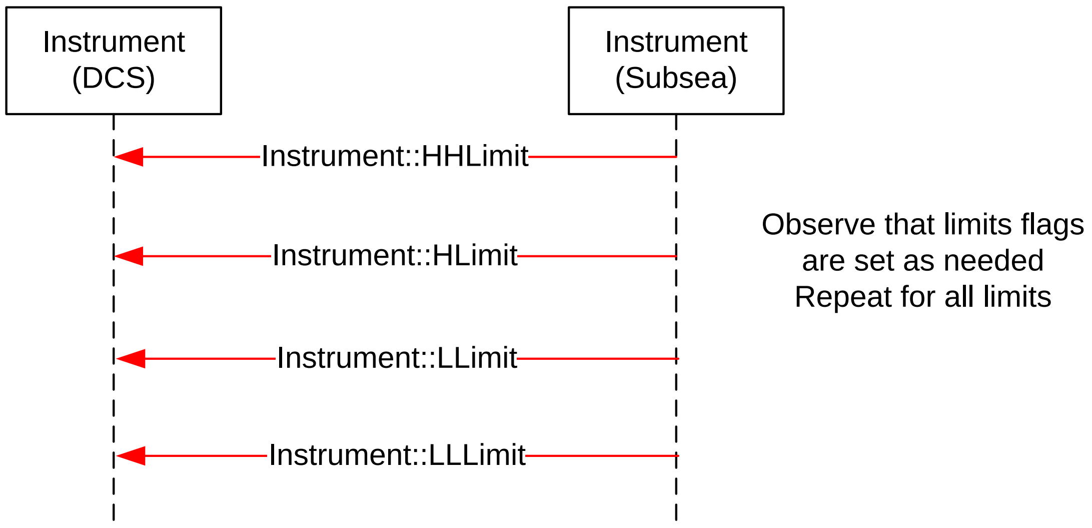

Figure 26 - Instrument Limits

A.4.4 Display Engineering units

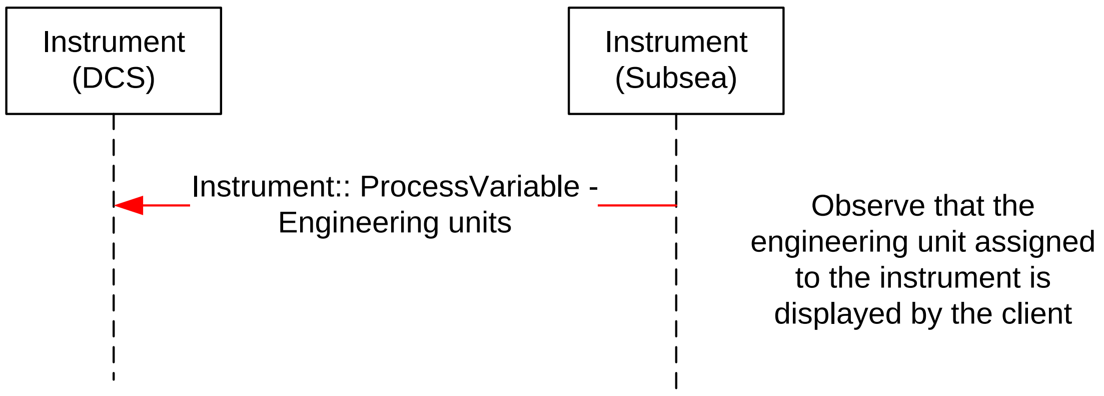
Instrument (DCS)
Instrument (Subsea)
Instrument:: ProcessVariable - Engineering units
Observe that the engineering unit assigned to the instrument is displayed by the client

Figure 27 - Instrument Engineering Units

A.5 MDIS Choke Object Sequence Diagrams

A.5.1 Overview

The following sequence diagrams indicate the intended SPCS and DCS interface operational steps. The sequence diagrams are used only to visualise different choke operations and to provide helpful information for implementation of the *MDISChokeObjectType* in OPC UA.

A.5.2 Move to Position – Success

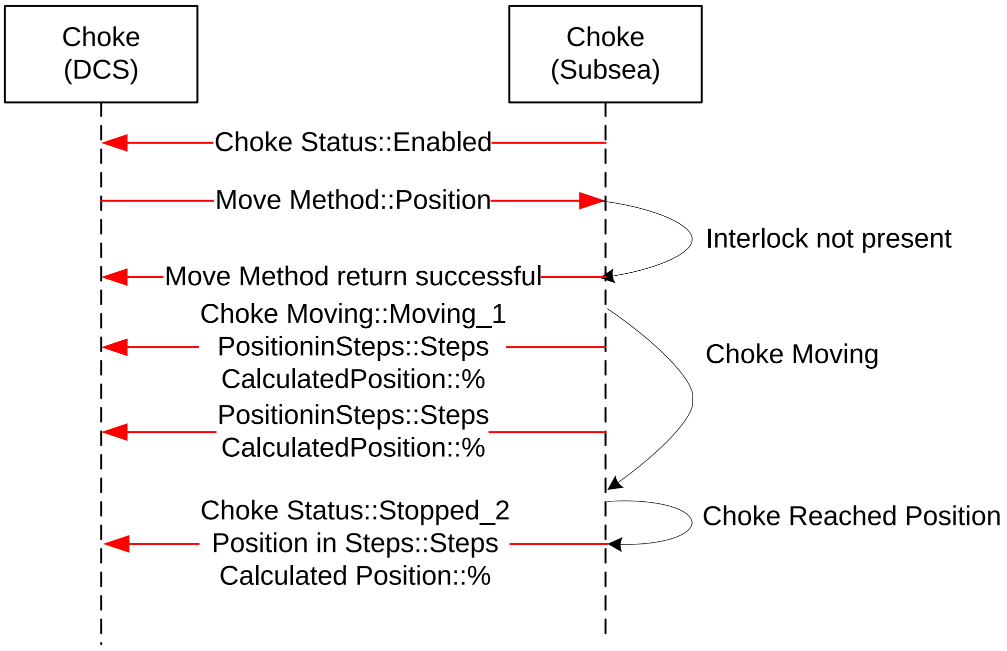

Figure 28 - Choke Move to Position

Sequence description; the above sequence details a successful execution of a Move to Position command [open or close] from the DCS to SPCS in addition to intermediate acknowledgements and states.

A.5.3 Move to Position – Fault

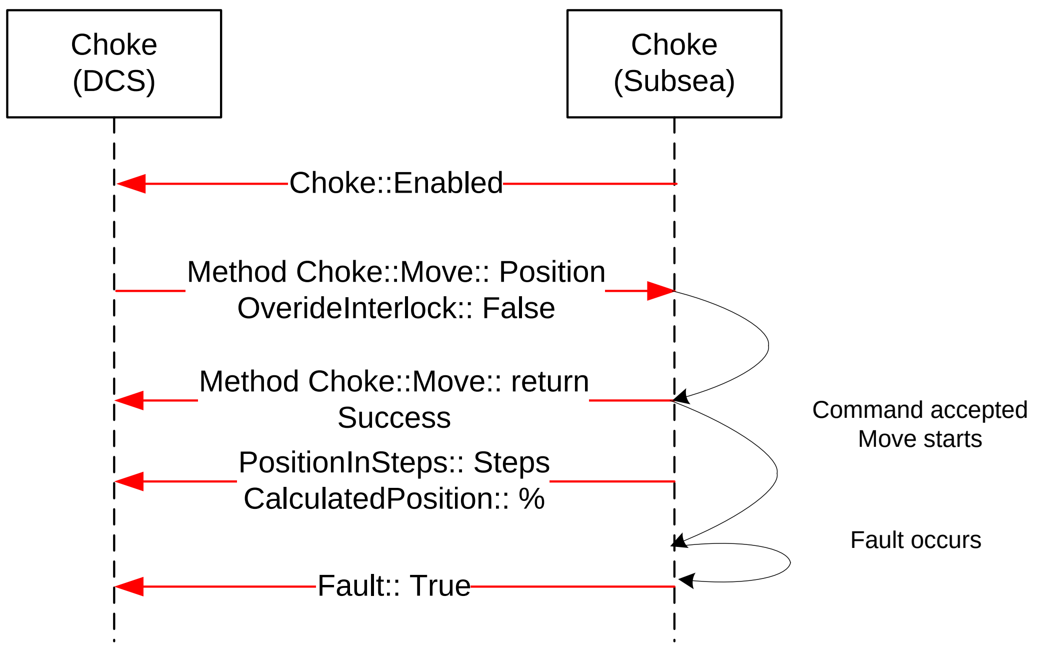

Figure 29 - Choke Move Fault

Sequence description; the above sequence details a Fault during the execution of a Move to Position command [open or close] from the DCS to SPCS in addition to intermediate states.

A.5.4 Move to Position – Failure, Interlock active

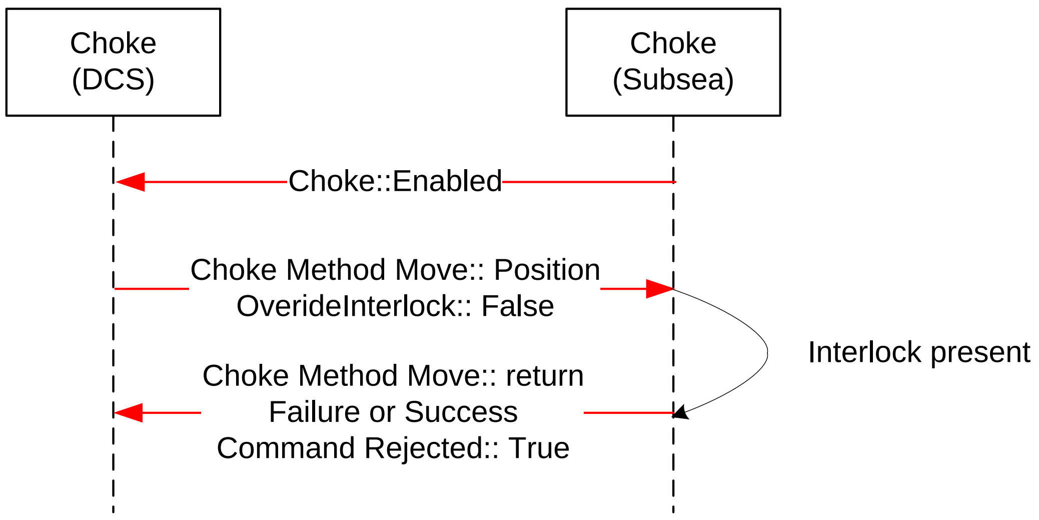

Figure 30 - Choke Move Interlocked

Sequence description; the above sequence details a failed execution of a Move to Position command [open or close] from the DCS to SPCS due to an Interlock being active.

A.5.5 Abort Choke (Position)

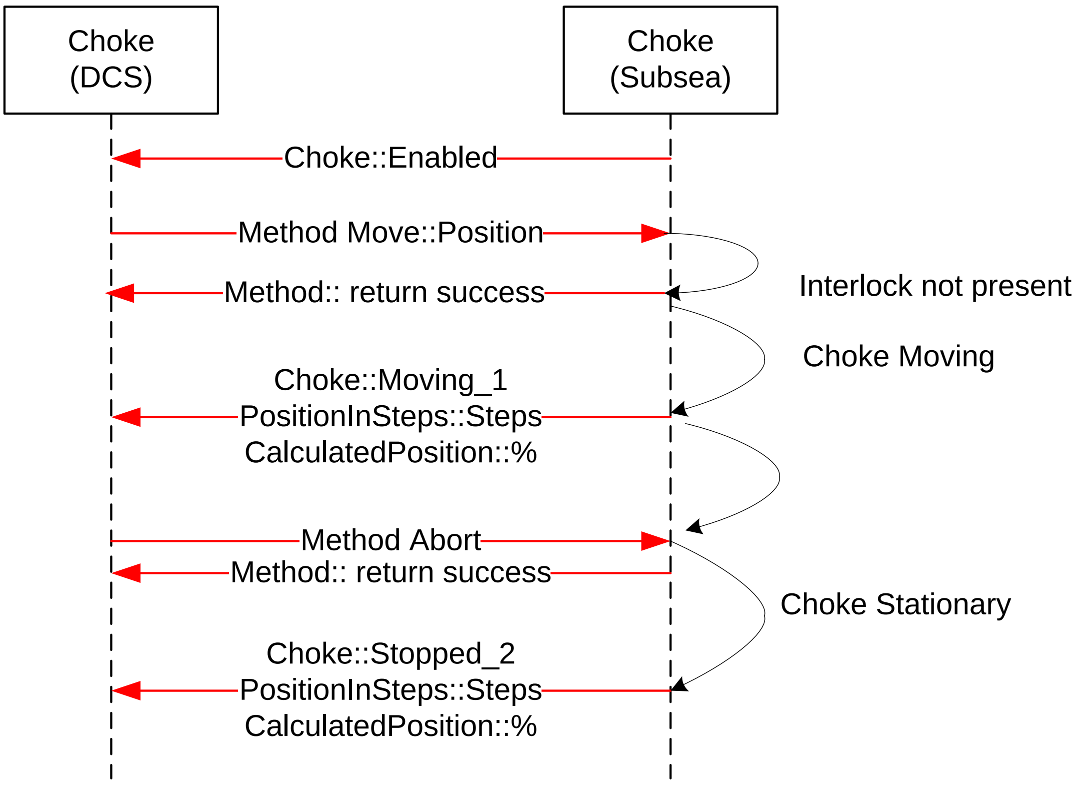

Figure 31 - Choke Move Abort

Sequence description; the above sequence details abort of a Move to Position command [open or close] from the DCS to SPCS.

A.5.6 Defeat / Override Interlock (Move)

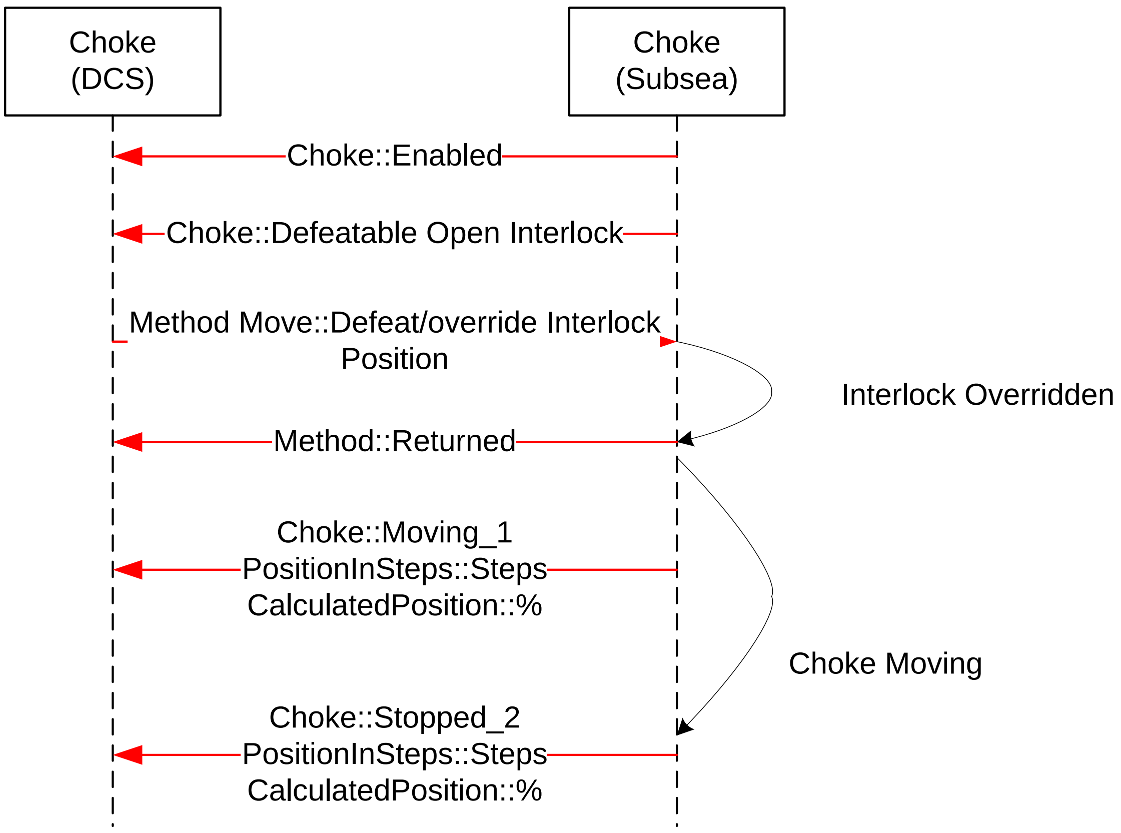
Choke (DCS)
Choke (Subsea)
Choke::Enabled
Choke::Defeatable Open Interlock
Method Move::Defeat/override Interlock Position
Interlock Overridden
Method::Returned
Choke::Moving_1 PositionInSteps::Steps CalculatedPosition::%
Choke Moving
Choke::Stopped_2 PositionInSteps::Steps CalculatedPosition::%

Figure 32 - Choke Move Interlock Override

A.5.7 Fault – No Move Operation

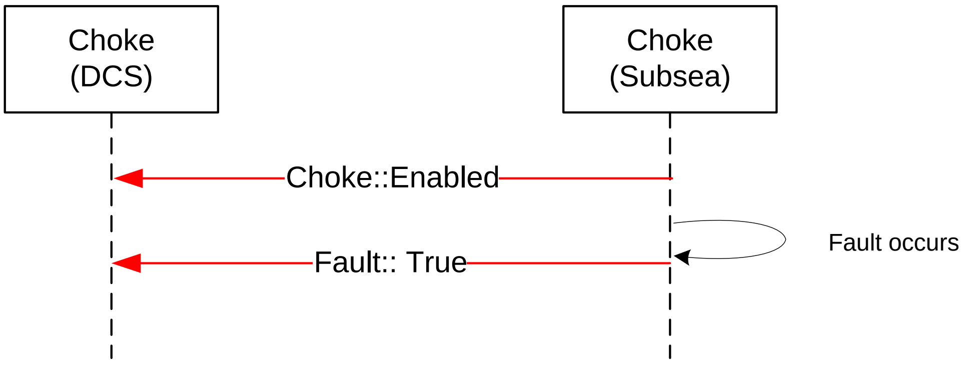

Figure 33 - Choke Fault

A.5.8 Step Open / Close – Success

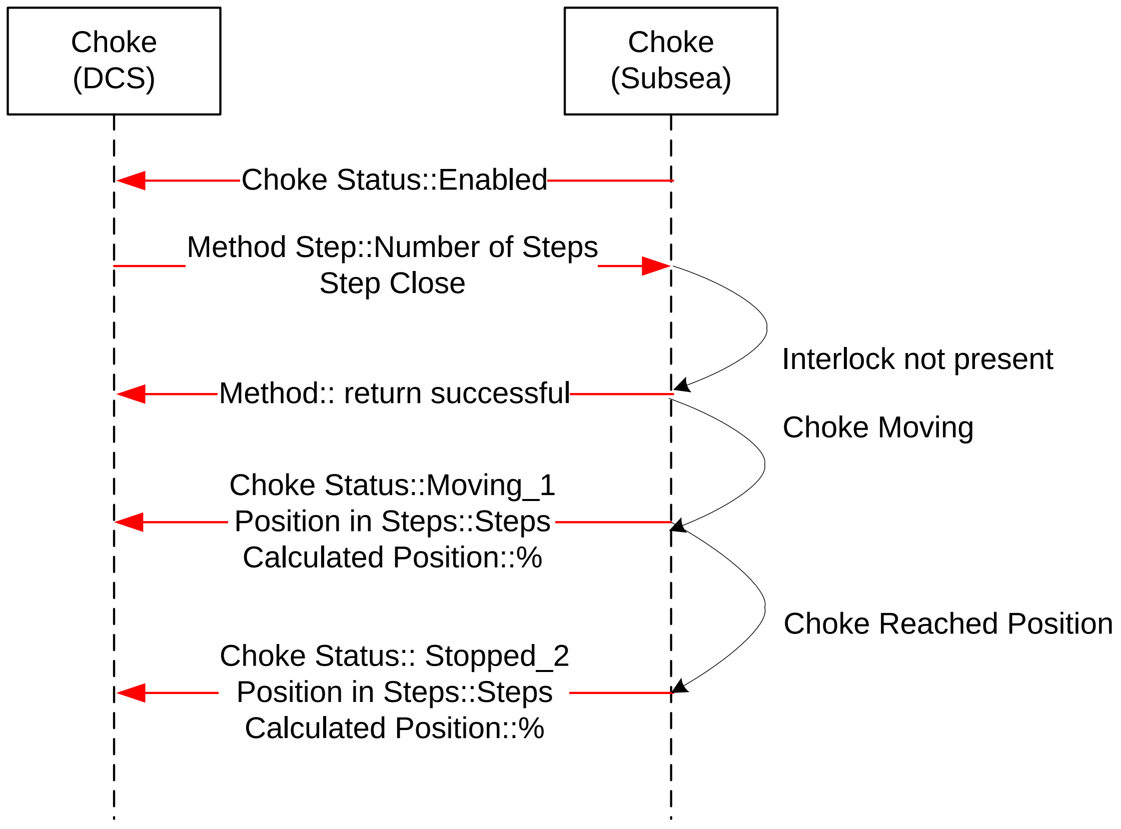

Figure 34 - Choke Step Success

Sequence description; the above sequence details a successful execution of a Step Open / Close command from the DCS to SPCS in addition to intermediate acknowledgements and states. The sequence diagram also includes information from instruments such as the Linear Variable Displacement (Differential) Transmitter (LVDT) to help illustrate what the actual information flow is.

A.5.9 Step Open / Close – Failure, choke fault

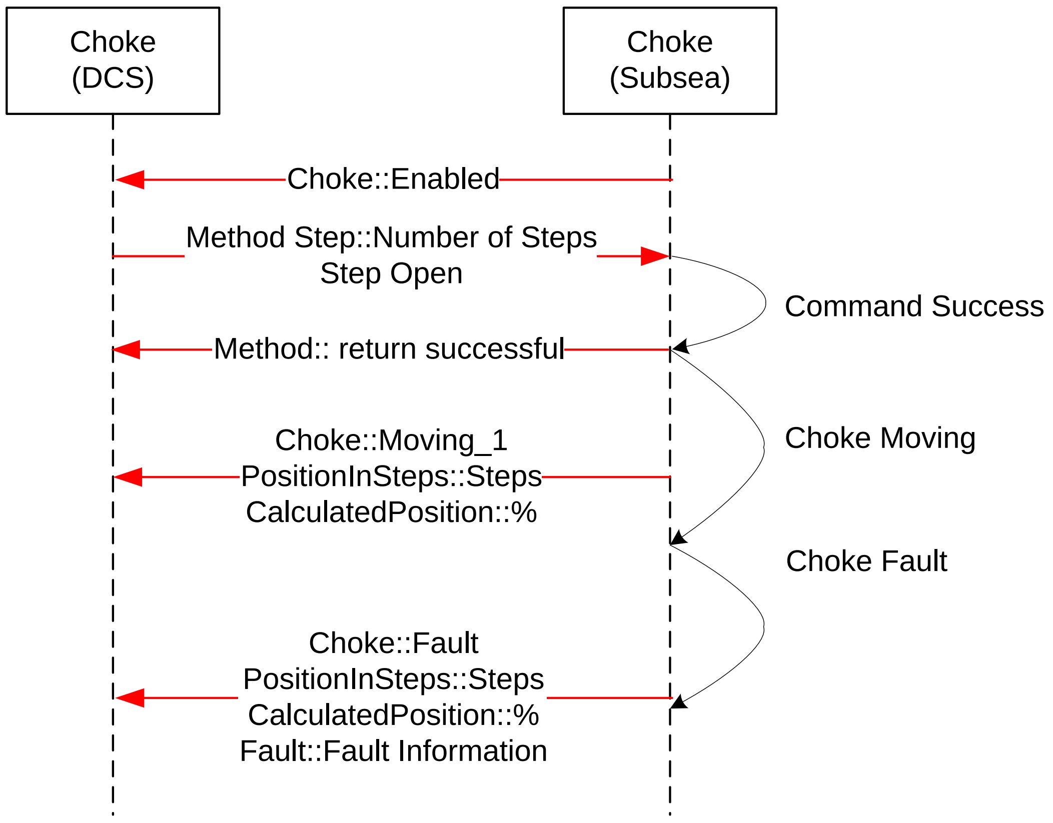
Choke (DCS)
Choke (Subsea)
Choke::Enabled
Method Step::Number of Steps Step Open
Command Success
Method:: return successful
Choke Moving
Choke::Moving_1 PositionInSteps::Steps CalculatedPosition::%
Choke Fault
Choke::Fault PositionInSteps::Steps CalculatedPosition::% Fault::Fault Information

Figure 35 - Choke Step Fault

Sequence description; the above sequence details a failed execution of a Step Open / Close command from the DCS to SPCS due to a choke fault.

A.5.10 Abort Choke (Step)

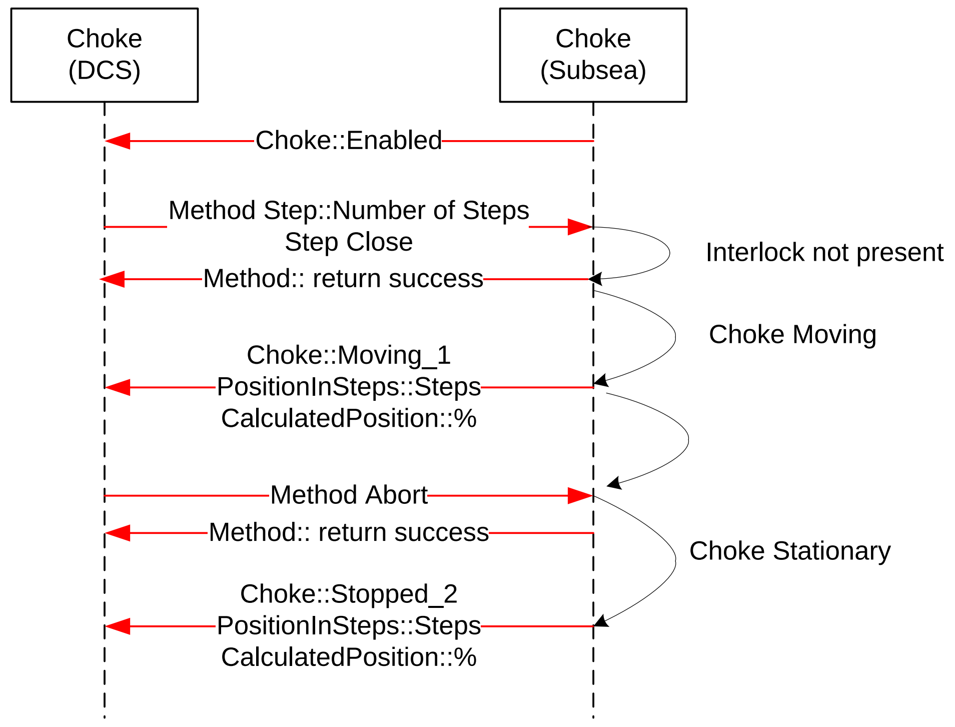
Choke (DCS)
Choke (Subsea)
Choke::Enabled
Method Step::Number of Steps Step Close
Interlock not present
Method:: return success
Choke Moving
Choke::Moving_1 PositionInSteps::Steps CalculatedPosition::%
Method Abort
Method:: return success
Choke Stationary
Choke::Stopped_2 PositionInSteps::Steps CalculatedPosition::%

Figure 36 - Choke Step Abort

Sequence description; the above sequence details a successful execution of a Choke Abort command from the DCS to SPCS in addition to intermediate acknowledgements and states.

A.5.11 Set Calculated Position

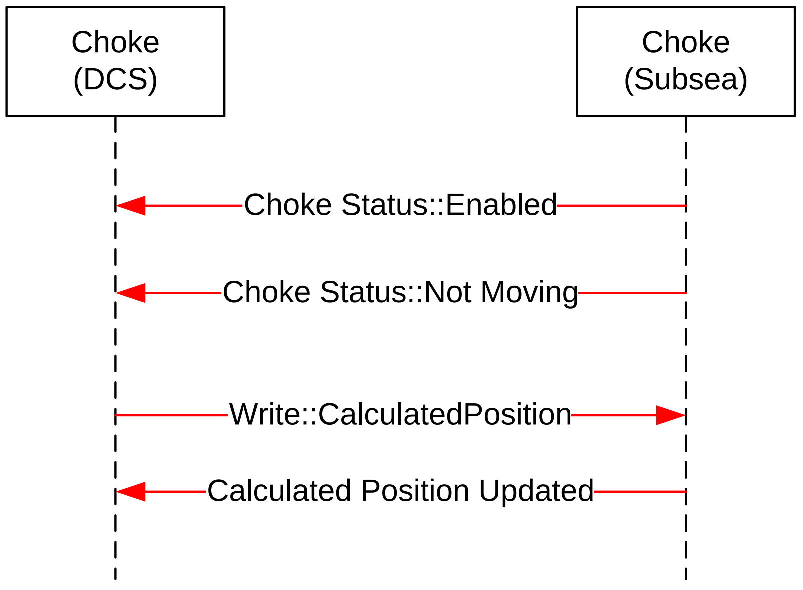

Figure 37 - Choke Set Position

Sequence description; the above sequence details a successful execution of a Set Calculated Position command from the DCS to SPCS.

A.5.12 Enable Disable Choke

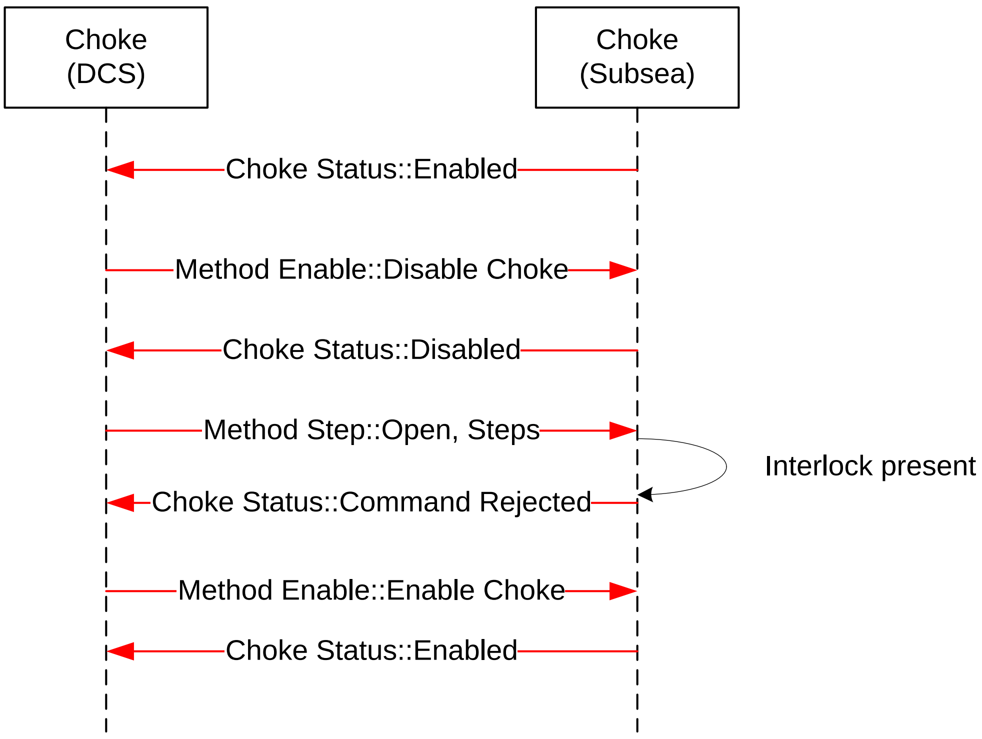

Figure 38 - Choke Enable / Disable

Sequence description; the above sequence details a successful execution of an Enable / Disable Choke from the DCS to SPCS in addition to intermediate acknowledgements and states.

A.5.13 Defeat / Override Interlock (Step)

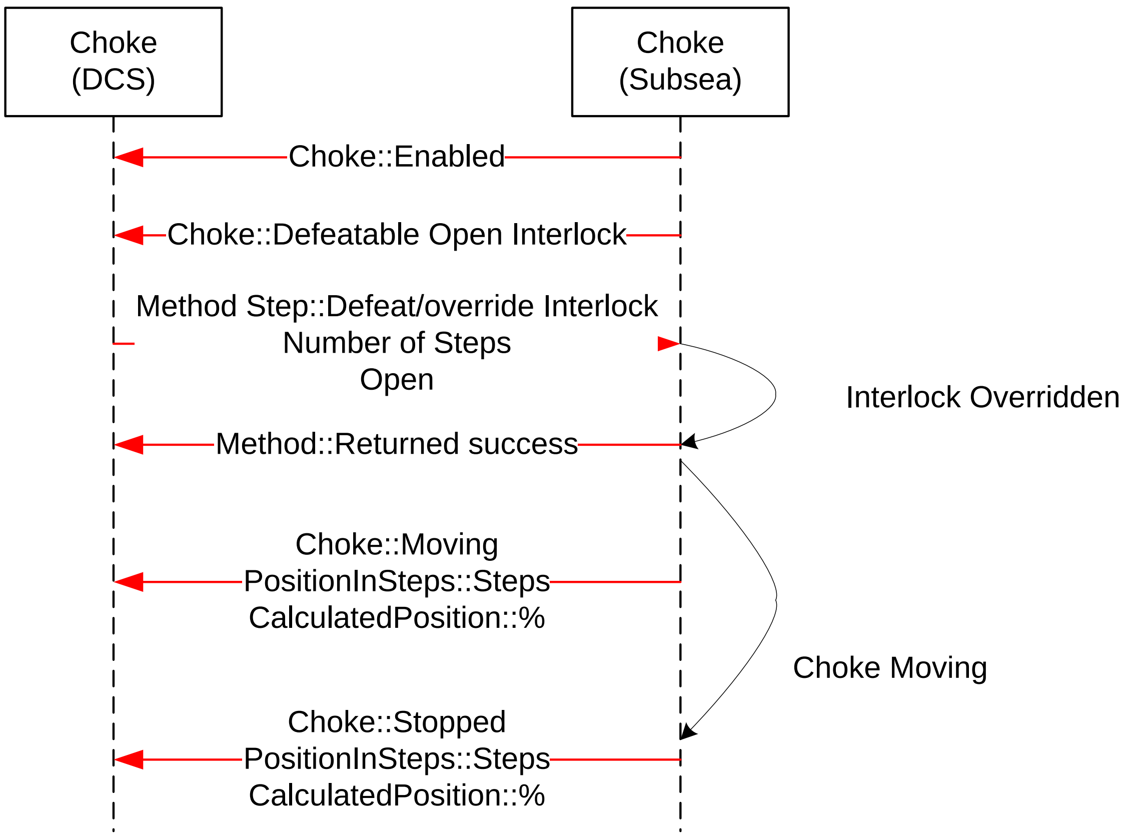

Figure 39 - Choke Step Interlock Override

Sequence description; the above sequence details a successful execution of a Defeat / Override Interlock Choke command from the DCS to SPCS in addition to intermediate acknowledgements and states

A.6 MDIS Valve Object Sequence Diagrams

A.6.1 Overview

The general functionality of the valve is to control the flow, in that it is either open and flowing or closed and not flowing. The *MDISValveObjectType* provides the information available for valves and provides access to control and management functionality in the valve. The following sequence diagrams indicate the intended subsea and DCS interface operational requirements and should be used in conjunction with the *MDISValveObjectType* generic properties. The final result described in the sequence diagram will be held until the next command is issued to the Valve, or until the state of the Valve changes (Fault or Interlock clears).

[Note: in most of the failure cases described below, the *Method* call should not have been made. But the error case is still described, since the Server still needs to be able to correctly handle the case where a *Client* sends an inappropriate command. Warning states do not affect commands.]

A.6.2 Valve command – Success

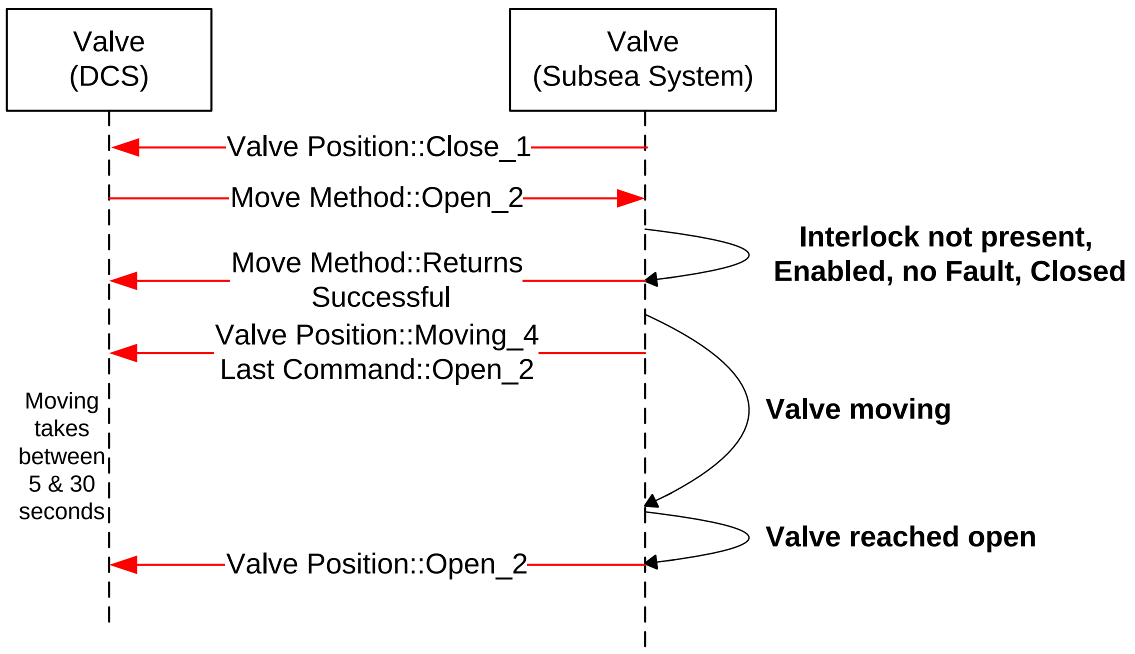
Valve (DCS)
Valve (Subsea System)
Valve Position::Close_1
Move Method::Open_2
Interlock not present, Enabled, no Fault, Closed
Move Method::Returns Successful
Valve Position::Moving_4 Last Command::Open_2
Valve moving
Moving takes between 5 & 30 seconds
Valve reached open
Valve Position::Open_2

Figure 40 - Valve command - success

Sequence description; the above sequence details a successful execution of a command [Open or Close] from the DCS to SPCS in addition to intermediate state changes.

A.6.3 Valve command – Overridden Interlock

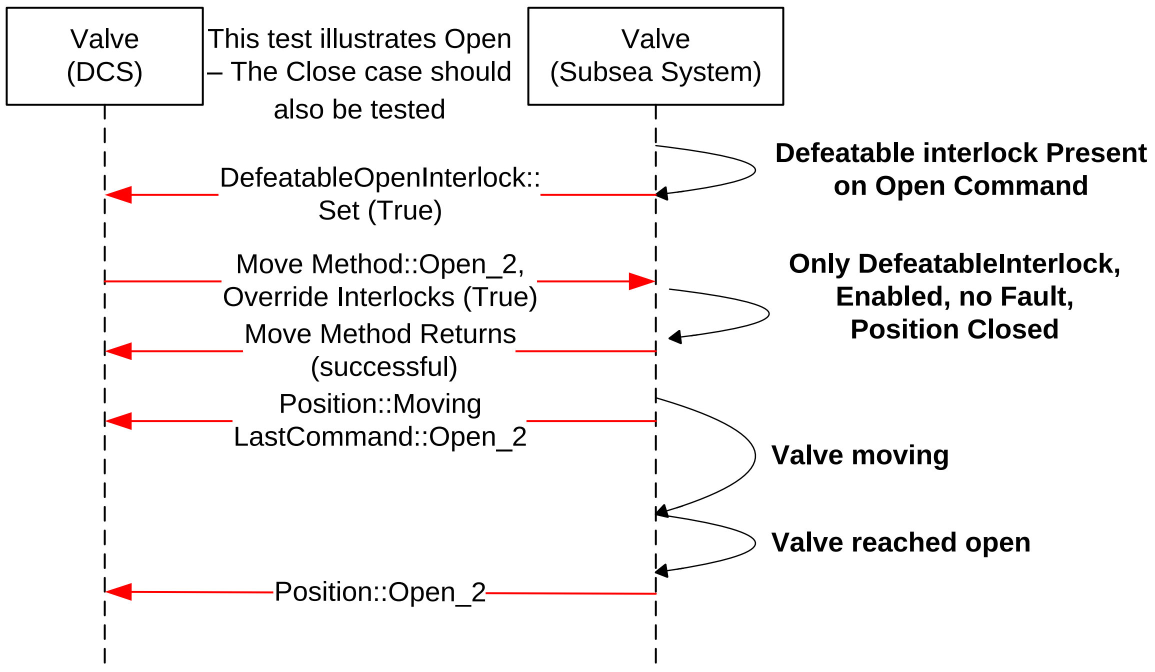
Valve (DCS)
This test illustrates Open – The Close case should also be tested
Valve (Subsea System)
Defeatable interlock Present on Open Command
DefeatableOpenInterlock:: Set (True)
Move Method::Open_2, Override Interlocks (True)
Only DefeatableInterlock, Enabled, no Fault, Position Closed
Move Method Returns (successful)
Position::Moving LastCommand::Open_2
Valve moving
Valve reached open
Position::Open_2

Figure 41 - Valve command – overridden Interlock

Sequence description; the above sequence details a successful execution of a command [Open or Close] with an interlock override active from the DCS to SPCS in addition to intermediate states changes. The Interlocks listed in the interlocks folder that are overridden are updated to reflect not interlocked.

A.6.4 Valve command –- Interlocked not overridden

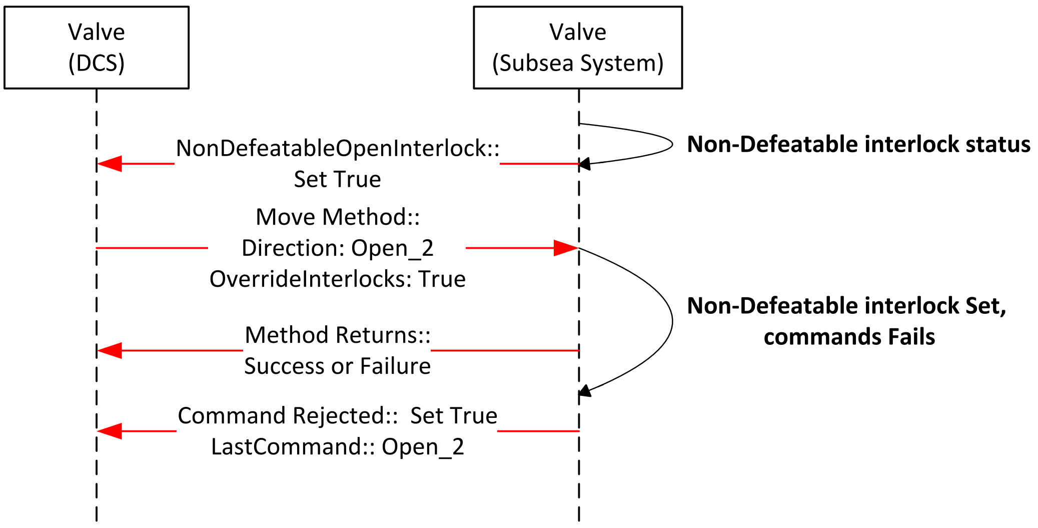
Valve (DCS)
Valve (Subsea System)
NonDefeatableOpenInterlock:: Set True
Non-Defeatable interlock status
Move Method:: Direction: Open_2 OverrideInterlocks: True
Non-Defeatable interlock Set, commands Fails
Method Returns:: Success or Failure
Command Rejected:: Set True LastCommand:: Open_2

Figure 42 - Valve command – Failed - Interlocked

Sequence description; the above sequence details the rejection to execute a command [open or close] owing to a non-defeatable interlock active status in the subsea system. The interlock is one reason a command may be rejected but there are others.

A.6.5 Valve command – Disabled

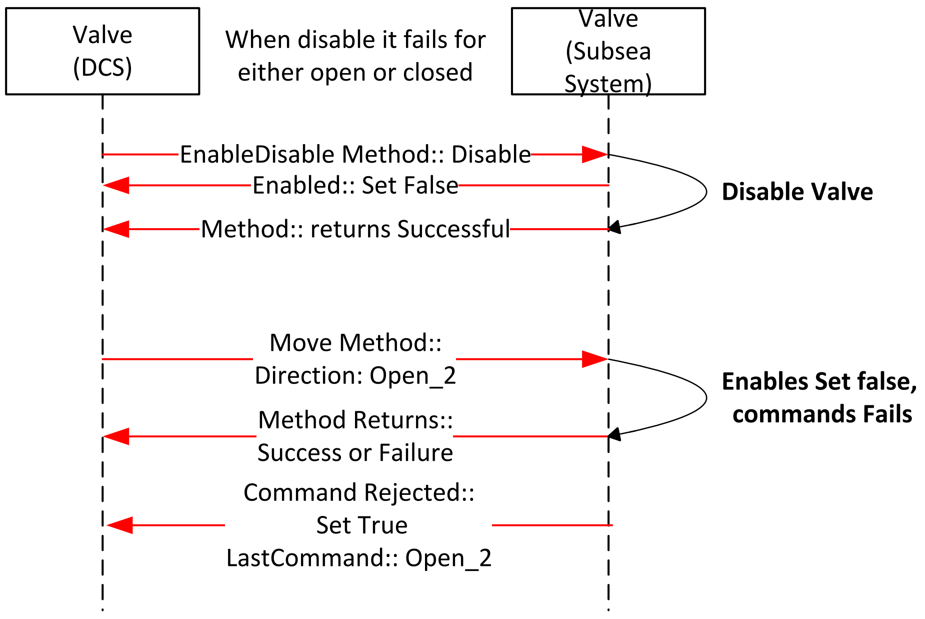
Valve (DCS)
When disable it fails for either open or closed
Valve (Subsea System)
EnableDisable Method:: Disable
Enabled:: Set False
Method:: returns Successful
Disable Valve
Move Method:: Direction: Open_2
Enables Set false, commands Fails
Method Returns:: Success or Failure
Command Rejected:: Set True LastCommand:: Open_2

Figure 43 - Valve command – Failed- Disabled

Sequence description; the above sequence details the rejection to execute a command [Open or Close] owing to the valve being disabled. The move *Method* may return successful or a failure depending on whether it knows if the valve is disabled.

A.6.6 Valve command – Failed – Fault case 1

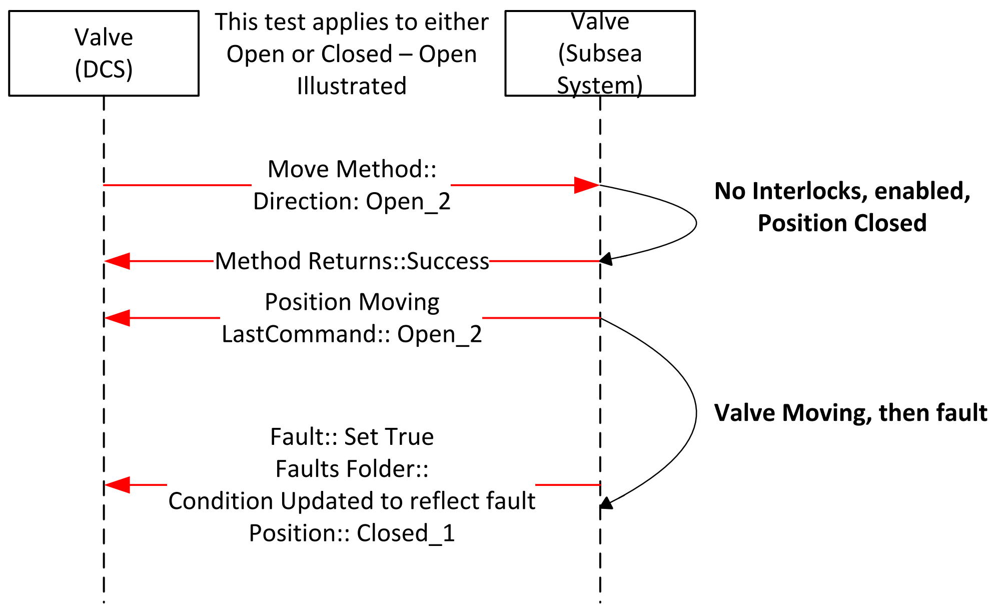
Valve (DCS)
This test applies to either Open or Closed – Open Illustrated
Valve (Subsea System)
Move Method:: Direction: Open_2
No Interlocks, enabled, Position Closed
Method Returns::Success
Position Moving LastCommand:: Open_2
Valve Moving, then fault
Fault:: Set True
Faults Folder:: Condition Updated to reflect fault
Position:: Closed_1

Figure 44 - Valve command – Failed - Fault

Sequence description; the above sequence is a failure of a command. The value fault maybe the result of the lack of action from the valve for a period of time greater that the OpenTimeDuration, it could also be some other fault that is reported from the subsea system.

A.6.7 Valve command – Failed – Fault case 2

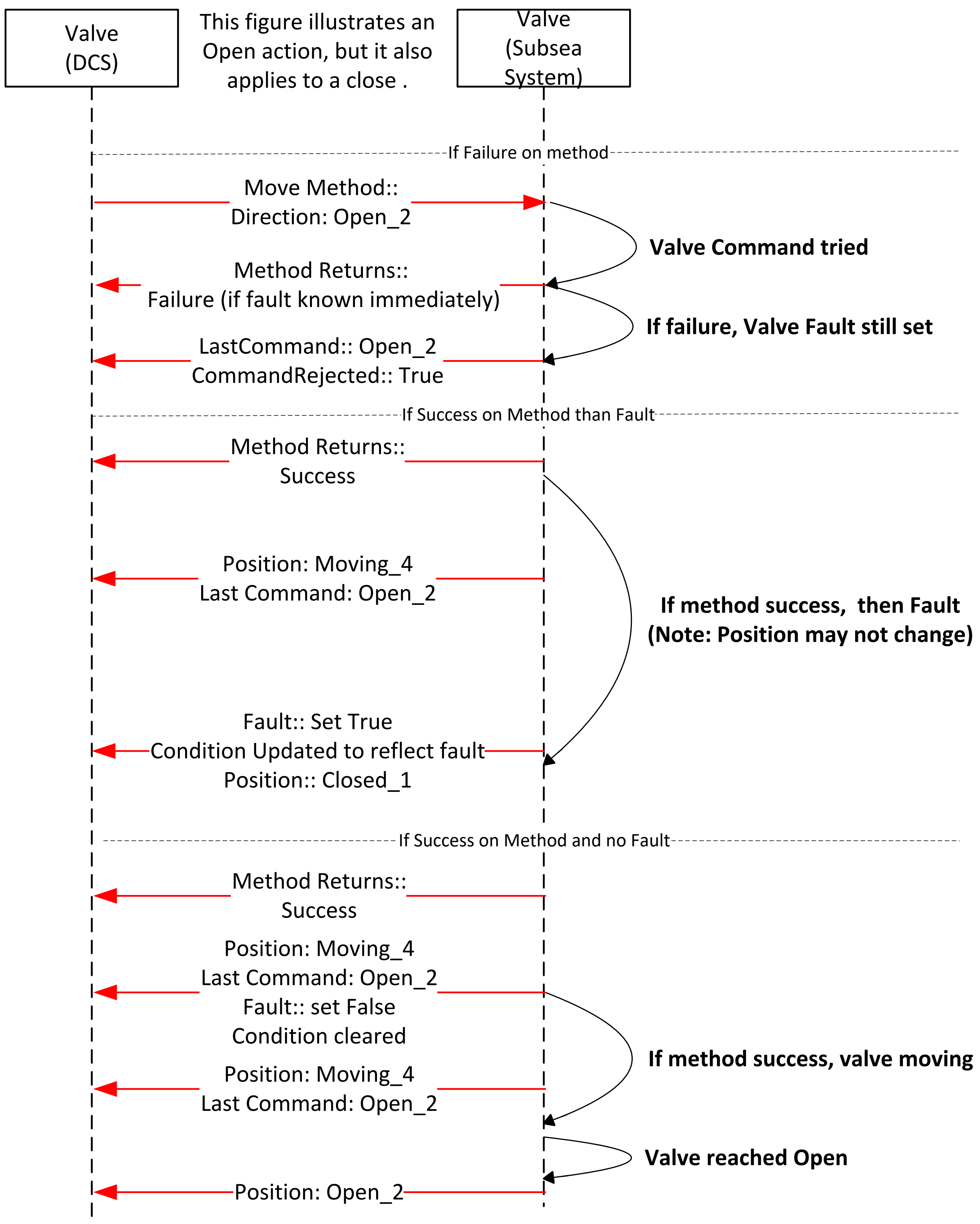
Valve (DCS)
This figure illustrates an Open action, but it also applies to a close .
Valve (Subsea System)
If Failure on method
Move Method:: Direction: Open_2
Valve Command tried
Method Returns:: Failure (if fault known immediately)
If failure, Valve Fault still set
LastCommand:: Open_2 CommandRejected:: True
If Success on Method than Fault
Method Returns:: Success
Position: Moving_4 Last Command: Open_2
If method success, then Fault (Note: Position may not change)
Fault:: Set True Condition Updated to reflect fault Position:: Closed_1
If Success on Method and no Fault
Method Returns:: Success
Position: Moving_4 Last Command: Open_2 Fault:: set False Condition cleared
If method success, valve moving
Position: Moving_4 Last Command: Open_2
Valve reached Open
Position: Open_2

Figure 45 - Valve command – Failed - Faulted

Sequence description; the above sequence details what the possible outcomes for moving a valve that has a fault set. The valve may immediately fault, may fault again after some time or the move may succeed. All outcomes are possible from a *Client* point of view. On some Servers only the last two may be possible.

A.6.8 Valve Signature Request – Completed

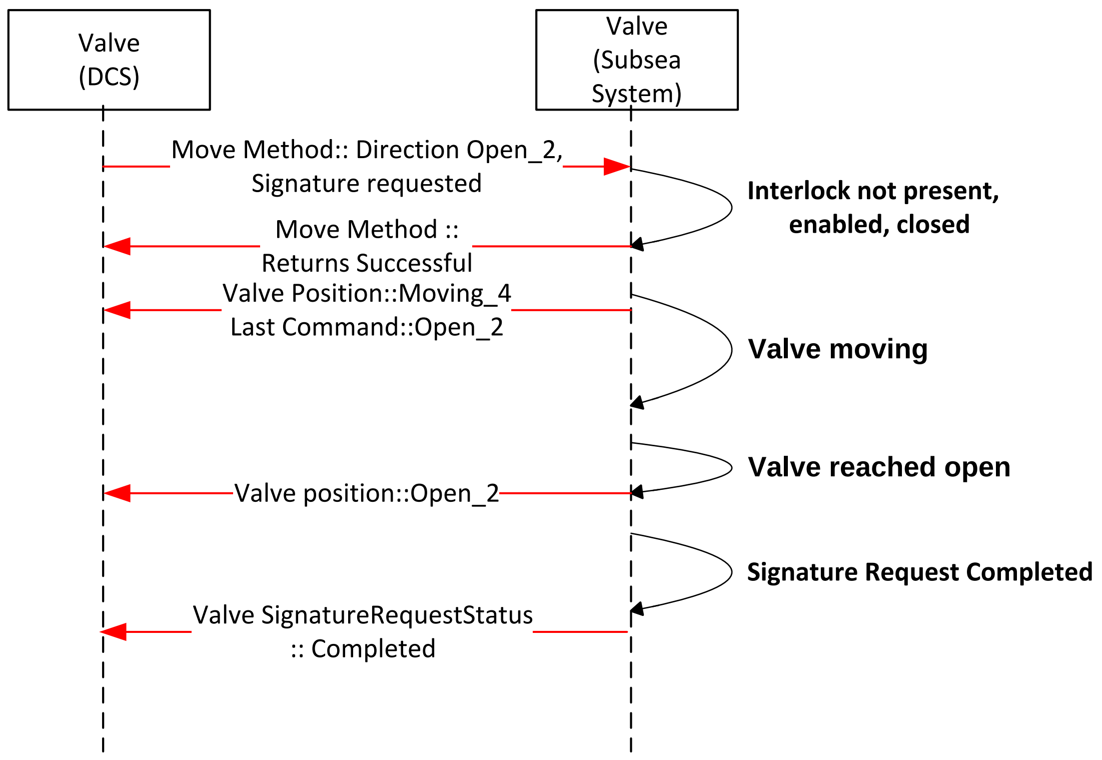
Valve (DCS)
Valve (Subsea System)
Move Method:: Direction Open_2, Signature requested
Interlock not present, enabled, closed
Move Method :: Returns Successful
Valve Position::Moving_4 Last Command::Open_2
Valve moving
Valve reached open
Valve position::Open_2
Signature Request Completed
Valve SignatureRequestStatus :: Completed

Figure 46 - Valve Profile Request – Completed

Sequence description; the above sequence details the valve signature request of a valve during operation [open or close] from the subsea system level. The valve may also report Failed or Not Available.

A.6.9 Valve command – Shutdown

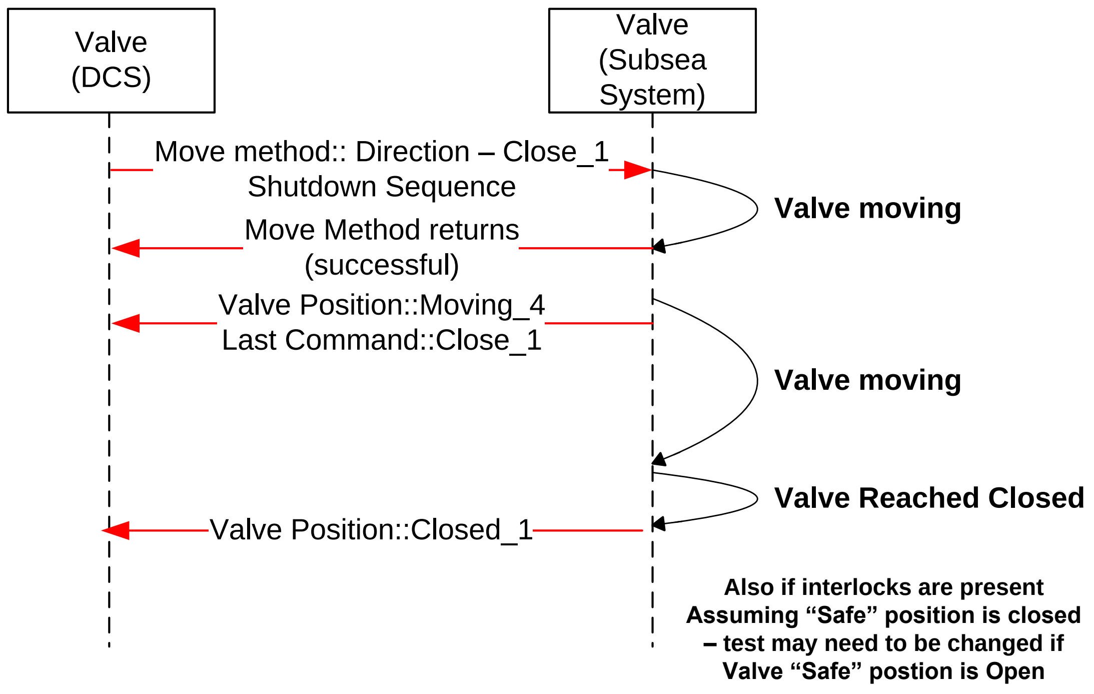
Valve (DCS)
Valve (Subsea System)
Move method:: Direction – Close_1 Shutdown Sequence
Valve moving
Move Method returns (successful)
Valve Position::Moving_4 Last Command::Close_1
Valve moving
Valve Reached Closed
Valve Position::Closed_1
Also if interlocks are present Assuming "Safe" position is closed – test may need to be changed if Valve "Safe" postion is Open

Figure 47 - Valve command – Shutdown

Sequence description; the above sequence details the execution of a Shutdown command [Open or Close]. The intermediate acknowledgements and states as applicable are indicated. The Shutdown command will attempt to override all interlocks, including any non-defeatable interlock if possible

Annex B Recommended Practice (Normative)

B.1 Introduction

This Recommended Practice provides guidance for use in specifying and implementing the OPC Unified Architecture (UA) for use as a standard communication interface for use between a Distributed Control System (DCS) and a Subsea Production Control System (SPCS) for control of subsea equipment.

Guidance for specifying minimum requirements when implementing MDIS are provided to ensure:

- Equipment compatibility when integrating the MCS or Subsea Gateway with the DCS.
- Safe and effective operation of equipment controlled by the MCS or Subsea Gateway from the DCS. Successful integration of the DCS with the MCS or Subsea Gateway during project execution.

B.2 Architecture Implementation

Physical architecture, defined in Section 4, should be specified. The location of the following minimum set of functionality, in the DCS or SPCS, is dependent on physical and “Operating Company” requirements and should be clearly specified prior to project execution:

- Data Arbitration
- Communication Channel Selection
- SEM Control Selection
- Process Interlocks
- Product Interlocks
- Shutdown Sequences
- Automated Control Sequences
- Valve Status Validation
- Choke Position Validation
- Interfacing with the HPU
- Interfacing with the topside chemical injection system
- Validation of Valve Profiles / Signatures

While functional logic has been prescribed by MDIS; implementation of specific functionality is not mandatory and should be specified on a project basis. As a minimum any MDIS implementation should ensure exchange of all required information required to safely line up, start up, operate and shutdown subsea equipment.

Interfaces with external interfaces with other third party systems (i.e. historians) and external interfaces between SPCS and DCS for use other than for control and monitoring of subsea equipment are not specified through the use of the MDIS OPC UA Companion Specification and should be specified on a project basis.

B.3 Security

Encryption or application level security requirements are not required on the protocol level.

It is recommended that all encryption or application level security requirements that are specified should be based on a network security assessment. If encryption on the protocol level is required, it is recommended to use encryption algorithms specified by OPC UA. Any advanced security features, if implemented, should have the option to be disabled and not adversely affect system performance.

No provision for user level security is considered for the MDIS Interface. The MDIS Interface is controller to controller communication and is not user restricted. All user level security should be implemented in the SPCS and DCS.

B.4 Performance

To maximise efficient use of bandwidth and performance, communication should be subscription based, by default.

The OPC UA interface is exception driven and a publishing interval of 2 seconds or less should be used between the SPCS and the DCS. Update rates from subsea to the SPCS is outside the scope of MDIS and can be much longer than 2 seconds. Maximum allowable update rates for individual controllers should be determined and based on system architecture and hardware capabilities.

Requirements should be developed on a project basis to provide sufficient bandwidth between the SPCS and DCS to ensure all commands, issued both during normal operation and shutdowns, are passed across the interface effectively.

Evaluation of communication performance and testing requirements should take into account any latency or communication limitations due to physical system architecture constraints between SPCS and the DCS. (i.e. radio communication).

B.5 Data Priority

Data prioritisation should be considered on a system level by each project and should take into consideration functional requirements for safe and effective operation of control of subsea equipment.

Data exchange across the interface should, unless otherwise specified be prioritised in the following manner:

- High Priority: Information required for process control.
- Low Priority: Information that is not required for process control (i.e. diagnostics, housekeeping).

B.6 Documentation

An interface specification should be developed between the SPCS vendor, DCS vendor and the “Operating Company” prior to the implementation of MDIS. The interface specification should provide the required fidelity to ensure all project specific requirements have been met. During the development of this specification it is recommended to review the project check list in Section B.8.

The SPCS vendor shall maintain revision control of the *UANodeSet* file that is being utilised by the project.

External interfaces (i.e. historians, condition monitoring systems, etc.) should also be specified however are outside the scope of this specification.

B.7 Interface Testing

The MDIS communication link shall be verified between the SPCS and DCS. Testing shall include both normal operation and failure scenarios. Redundancy and performance requirements should be tested under full load (i.e. shutdown conditions). Time synchronisation should be verified.

Clients and *Servers* that have passed independent MDIS certification should require less integration testing. Integration testing should be focused on the project configuration rather than base MDIS functionality. It is recommended prior to any integration testing that the interface specification, defined in Section B.6, and OPC UA *UANodeSet* files be finalised and agreed between SPCS and DCS vendors as early as possible. Integration testing should only be attempted when the SPCS and DCS software has reached an acceptable level of maturity.

It is recommended that, during project execution, preliminary integration testing be performed to validate the data exchange across the MDIS Interface off critical path, prior to full onsite integration testing. In addition to validating data exchange, this testing should allow for verification of graphical displays, redundancy and data prioritisation. Validation of performance and full functionality, including shutdowns, sequences and interlocks, should be performed during full onsite integration testing using the hardware that will be supplied to the project.

B.8 Project Check List

B.8.1 Introduction

This Checklist provides general guidance for use in tender and development cycles for implementing the MDIS Interface between a Distributed Control System (DCS) and a Subsea Production Control System (SPCS) for control of subsea equipment.

B.8.2 Front End Engineering Design (FEED)

The following checklist is ordered by decisions and activities that should be defined during clarification and development of an MDIS interface specification interface.

Table 96 – Checklist – FEED Scope

No	Item Description	Comments	√
1.	The Operator, DCS vendor and SPCS vendor should assign dedicated persons to develop requirements for the interface.	Knowledge of latest MDIS Specification is recommended.	
2.	Operator to schedule Kick-off or Workshop with DCS vendor and SPCS vendor to discuss checklist items.		
3.	Agree on system architecture. The Operator shall decide which MDIS architecture (Integrated, Interfaced or Other) is applicable. The definition of architecture should document where all functional logic should reside, as per Section B.2.		
4.	Create an overview drawing of the hardware and network architecture. Decide on Interface setting (IP ranges and class). Define hardware components on both systems that are related for the interface.		
5.	DCS vendor to provide OPC Foundation accredited Test Lab certification report (OPC UA Client), if available, to the Operator.		
6.	SPCS vendor to provide OPC Foundation accredited Test Lab certification report (OPC UA Server), if available, to the Operator.		
7.	Select responsible Party (DCS or SPCS) for implementing shutdown Sequences.		
8.	Select responsible Party (DCS or SPCS) for implementing automated Sequences.		

No	Item Description	Comments	√
9.	Select responsible Party (DCS or SPCS) for implementing Product Interlocks.		
10.	Select responsible Party (DCS or SPCS) for implementing Process Interlocks.		
11.	Select responsible Party (DCS or SPCS) for creation and managing the Product Alarms/Warnings. Specify detail level of product alarms (summary alarm or specific information)	SPCS Vendor should provide diagnostic information.	
12.	Select responsible Party (DCS or SPCS) for creation and managing Process Alarms/Warnings.		
13.	Define where information from redundant physical measurements is arbitrated.	Example: Should the SPCS or DCS be responsible for arbitrating information from redundant pressure / temperature transmitters that are measuring the same process variable?	
14.	Define where management of redundancy in subsea equipment is arbitrated.	Example: Should the SPCS or DCS be responsible for assigning which SEM is used to command valve actuation? Should the SPCS or DCS be responsible for assigning a “primary” or “backup” SEM?	
15.	Define where redundant information due to multiple communication channels is arbitrated.	Example: Dependent on system design a single process sensor value might be available through various communication channels. Should the SPCS or DCS be responsible for determining which value is used for display to the operator and used as input to interlocks and automated sequences?	
16.	Define the instance model for redundant systems	Example: Is the instance model on each redundant channel identical or do the instance models represent separate and redundant pathways to subsea equipment? See section 11 for more details.	
17.	Make a list of all MDIS Object types that will utilized on the project.	Object types that are not utilized on the project can be skipped from following optional item selection.	

No	Item Description	Comments	√
18.	Operator, DCS and SPCS to select required optional References and Methods for all MDIS objects: - MDIS Discrete Instrument - MDIS DiscreteOut Instrument - MDIS Digital Instrument - MDIS DigitalOut Instrument - MDIS Instrument - MDIS InstrumentOut - MDIS Valve - MDIS Choke - Etc.	It is recommended to start discussions assuming all objects use all mandatory and optional References and Methods and remove items that are not project relevant.	
19.	Supported arguments for the MDIS Valve and MDIS Choke Move-Method calls should be documented.	Available Move-Method calls are defined in the MDIS Valve and MDIS Choke objects.	
20.	Faults and warning codes warning provided in SPCS system should be defined. Specific actions to be initiated by the DCS in the event of a fault or warning code should be documented.		
21.	Agree on standardised methodology for naming of *Objects* used on the interface (Hierarchy / Field structure)	Project specific equipment specific tag identifiers can be used on the interface to name equipment.	
22.	SPCS Vendor to prepare list of available MDIS Aggregated Objects and Vendor specific Subtypes of MDIS Objects.	This list should provide sufficient detail to allow development of software by both the DCS and SPCS provider.	
23.	The structure of address space across the interface (folder structure or MDIS Aggregated object) should be defined.	Address space structure should be clearly documented by the SPCS vendor and be reviewed by the DCS vendor to ensure no incompatibility.	
24.	A method for time Synchronization shall be defined.		
25.	FEED work should be documented to clarify expectations during project execution.		

B.8.3 Project Execution

The following checklist documents activities that should be considered during project execution.

Table 97 - Checklist - Project Execution

No	Item Description	Comments	√
1.	SPCS Vendor should create a project specific MDIS Interface specification for DCS.	This MDIS Interface specification should be based on documented FEED discussions.	

No	Item Description	Comments	√
2.	SPCS Vendor to schedule date for delivery of MDIS NodesetFile(s) for DCS Vendor.	More than one date might be scheduled, where each date would provide a NodesetFile that includes additional object definitions. For example, an initial NodesetFile might contain just the first prototype of a well, and subsequent NodesetFiles might contain a more detailed well and then multiple instances of the well. Also, some Objects might include additional optional items as the well model is finalized.	
3.	Schedule date for first MDIS interface connection Test. SPCS vendor, DCS vendor and Operator should agree on test cases for first MDIS interconnection test.	First interconnection test should cover test cases such as redundancy, time synchronization, performance, etc.	
4.	Check possibility/availability of an online SPCS OPC UA MDIS Server to support and accelerate interface development.	Operator to support/provide infrastructure (e.g. use own infrastructure or cloud systems) if link between SPCS and DCS Vendor cannot be established.	
5.	DCS Vendor to prepare Interface Test procedure. Schedule review cycle for Operator and SPCS vendor.		
6.	Schedule Date for final MDIS interface connection Test (full project configuration, including sequences and interlocks, etc.).		

B.8.4 Closeout

The following Checklist documents activities that should be completed complete as part of a project closeout.

Table 98 - Checklist - Project Closeout

No	Item Description	Comments	√
1.	Feedback lessons learned and improvements from project execution to MDIS working group.	Feedback should be provided from all parties (Operator, DCS, SPCS). Feedback could include discussion of specification features, feedback on this checklist, future features that could be added to specification etc.	

Annex C Alternative MDIS Applications (Normative)

C.1 Introduction

OPC UA, through the MDIS Companion Specification, has been specified as a controls interface between the MCS and DCS. This normative guideline provides details on other potential applications of the MDIS standard and additions to standard MDIS implementations that are outside the original scope of MDIS. Concepts covered include:

a.) Application of OPC UA and standard objects and concepts defined within the MDIS Companion Specification as "read only" interfaces for historian applications.

b.) Additions to OPC UA MDIS implementations that can facilitate data transfer that are not specifically control related.

C.2 Read Only Interface

The MCS can provide "read only" interfaces to facilitate data to external databases, as specified on a project basis. In these cases, OPC UA can be specified and use of standard objects and concepts taken from the MDIS standard can be utilized based on project specific requirements.

To use the OPC UA interface as a "read only" interface it is recommended to that a Server be able to indicate that all writeable points (by definition have access restriction that denies write for all users and that no writeable ObjectType instance will exist in the address space. For all methods that allow writing, the user access rights should indicate not executable. The Server should be configurable to support this "read only" configuration. The server should not be able to be changed from this configuration without a restart.

C.3 Signature File Transfer

Valve signatures are files that contain high sample rate pressure profiles recorded during individual valve operations subsea exposed by the subsea system. There are cases where valve signatures are required to be transferred from the SPCS to either the DCS or external historian interfaces.

A *Server* that supports transferring the valve signature via OPC UA shall expose a *FileObject* as part of the Valve Object Instance via the *HasSignature ReferenceType*. The valve Signature will also be provided in the MDIS folder under the *MDISInformation Object*. File formats and specific information captured within the valve signature files are outside the scope of this specification and should be specified on a project basis.

The items required are provided as part the individual object types. This functionality is optional and covered in a separate profile.