OPC Unified Architecture

for

Analyser Devices

Companion Specification

Release 1.1a

Jan 9, 2015

Specification Type	Industry Standard Specification	Comments:	
Title:	OPC Unified Architecture Analyser Devices	Date:	Jan 9, 2015
Version:	Release 1.1a	Software	MS-Word
		Source:	OPC UA For Analyser Devices 1.1a Companion Specification.doc
Author:	OPC Foundation	Status:	Release

CONTENTS

FIGURES

TABLES

OPC FOUNDATION

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 to develop a standard interface to facilitate the development of applications by multiple vendors that shall inter-operate seamlessly together.

GENERAL PROVISIONS

Should any provision of this Agreement be held to be void, invalid, unenforceable or illegal by a court, the validity and enforceability of the other provisions shall not be affected thereby.

This Agreement shall be governed by and construed under the laws of the State of Minnesota, excluding its choice or law rules.

This Agreement embodies the entire understanding between the parties with respect to, and supersedes any prior understanding or agreement (oral or written) relating to, this specification.

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 1.1a Highlights

This revision contains only two typos corrections in links in Annex H.

Revision 1.1 Highlights

This revision contains minor modifications simplify implementation, fix typos, synchronize with latest OPC UA and DI specifications.

The following table includes the main changes.

Summary	Resolution
Add support for large configuration	Defined an optional ConfigData File object and associated rules allowing an external to set the ConfigData larger than a single OPC UA message.
Add support for pluggable AnalyserChannel and Stream	Allow AnalyserDevice to have 0..n AnalyserChannel and AnalyserChannel to have 0..n Stream.
Simplify implementation of state machines	Now, all methods are only part of MethodSet, they no longer need to be part of objects themselves neither state machines.
Synchronize with the DI specification v1.1 release	Use same conventions as DI specification v1.1 like DeviceHealth enumeration and profiles
Synchronize with OPC UA v1.02	Use new types defined in OPC UA Part 8.
Offset	The time between the start sample extraction and the start of the analysis is now reflected in the optional AcquisitionData.Offset parameter.
Time management	Time stamp of the acquisition data is now the time of the extraction of the sample.

1 Scope

This specification is an extension of the overall OPC Unified Architecture specification series and defines the information model associated with analytical devices (analysers). The model described in this specification is intended to provide a unified view of analysers irrespective of the underlying device protocols.

2 Reference documents

[ISA-88] ANSI/ISA 88.01-1995 Batch Control Part 1: Models and terminology

[ISA-88 TR] ANSI/ISA TR 88.02-2008 Machine and Unit States Technical Report

[NE-107] NAMUR Recommendation, Self-Monitoring and Diagnosis of Field Devices.

[UA-DI] OPC UA for Devices 1.1 Companion Specification.

http://www.opcfoundation.org/UA/UADevices

[UA Part 1] OPC UA Specification: Part 1 – Concepts.

http://www.opcfoundation.org/UA/Part1/

[UA Part 3] OPC UA Specification: Part 3 – Address Space Model.

http://www.opcfoundation.org/UA/Part3/

[UA Part 4] OPC UA Specification: Part 4 – Services.

http://www.opcfoundation.org/UA/Part4/

[UA Part 5] OPC UA Specification: Part 5 – Information Model.

http://www.opcfoundation.org/UA/Part5/

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

[UA Part 8] OPC UA Specification: Part 8 – Data Access.

http://www.opcfoundation.org/UA/Part8/

3 Terms, definitions, and abbreviations

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in [UA Part 1], [UA Part 3], [UA Part 4], [UA Part 5], [UA Part 7], [UA Part 8], and [UA-DI] as well as the following apply.

3.1.1
Accessory

a physical device which can be mounted on the Analyser or Analyser Channel to enhance its behaviour or operation.

NOTE: Examples of accessories are: vial holder, filter wheel, auger, and heater.

3.1.2
Accessory Slot

a physical location on the Analyser or Analyser Channel where an Accessory can be attached.

3.1.3
Analyser Device

a device comprised of one or more analyser channels with a single address space which has its own configuration, status and control.

3.1.4
Analyser Channel

a subset of an Analyser that represents a specific sensing port and associated data, which includes scaled data (e.g. spectrum), configuration, status and control.

3.1.5
Analyser Client

an OPC UA *Client*, which is aware of the ADI *Information Model*.

3.1.6
Analyser Configuration

a set of values of all *Parameter*s that when set, put the analyser in a well defined state.

3.1.7
Analyser Model

a description of a mathematical process and associated information to convert raw data into scaled data.

3.1.8
Analyser Server

an OPC UA *Server*, which implements the ADI *Information Model*.

3.1.9
Calibration

one or more acquisitions using reference samples in order to determine the factors used to convert analyser raw data to scaled data.

3.1.10
Chemometric Model

a description of a mathematical process and associated information to convert scaled data into one or more process values (process data).

3.1.11
Chromatographic Application

a defined series of hardware, valves, columns, and detectors, to produce an chromatographic result on a requested process stream analysis.

3.1.12
Parameter

a specialization of *Parameter* defined in [UA-DI] for *AnalyserDevice, AnalyserChannel, AccessorySlot, Accessory* or *Stream* and used to configure or publish information about the analytical device or its components.

NOTE: All Parameters described in this specification are represented by OPC UA Variables.

3.1.13
Process Data

Data generated from scaled data by applying a chemometric model.

NOTE: Process data is typically represented as a scalar value or a set of scalar values and it is often used for process control. Examples of process data are: concentration, moisture and hardness.

3.1.14
Raw Data

Data generated by an analyser representing an actual measurement but without any meaningful units.

NOTE: Raw data is typically represented as an array of numbers. Examples of raw data are: raw spectrum, chromatogram and particle size beam count. Typically, this data is not directly consumed by a Client.

3.1.15
Sampling point

a physical interface point on the process where the process is monitored. Certain analysers perform in-place, non-destructive measurements whereas others extract a sample.

3.1.16
Scaled Data

Data generated from raw data and representing an actual measurement expressed in meaningful units.

NOTE: Scaled data is typically represented as an array of numbers. Examples of scaled data are: absorbance, scatter intensity.

3.1.17
Stream

a mapping between an AnalyserChannel and the process sampling points.

NOTE: One AnalyserChannel can handle one or more sampling points, which means that an AnalyserChannel can be associated with one or more Streams.

3.1.18
Validation

one or more acquisitions using reference samples to demonstrate that the results provided by the analyser are still within the acceptable ranges.

3.2 Abbreviations and symbols

ADI	Analyser Device Integration
ATR	Attenuated Total Reflectance
DA	Data Access
DCS	Distributed Control System
DI	Device Integration
HMI	Human Machine Interface
LIMS	Laboratory Information Management System
OEM	Original Equipment Manufacturer
OPC-ADI	Namespace of the Unified Architecture Analyser Device Interface *Information Model*
OPC-DI	Namespace of the Unified Architecture Devices *Information Model*
OPC-UA	Namespace of the Unified Architecture *Information Model*
UA	Unified Architecture

3.3 Naming convention

Instances are referred to using the same identifiers as their type definition without *Type* suffix.

Identifiers described as a name enclosed in angle brackets e.g. <ParameterIdentifier> or <GroupIdentifier> represent identifiers assigned by the Analyser Server and not explicitly defined by this specification.

4 Concepts

4.1 General

This specification defines an *Information Model* for analysers. This *Information Model* is also referred to as the ADI *Information Model*. Analysers can be further refined into various groups such as light spectrometers, particle size monitoring systems, imaging particle size monitoring systems, acoustic spectrometers, mass spectrometers, chromatographs, Imaging systems and nuclear magnetic resonance spectrometers. These groups can be extended and each group can also be further divided. The requirements for all of these groups of *analysers* can vary, but this specification defines an *Information Model* that can be applied to all groups of *analysers.*

OEM integrators often build specialized analytical devices, e.g. octane monitor, by combining several off-the-shelf analysers and accessories. That kind of compound analytical device can be treated as yet another type of *Analyser* to which this *Information Model* applies.

4.2 Overview

The object model that describes analysers is separated into a definition of *AnalyserDevice, AnalyserChannel, Stream, Accessory* and *AccessorySlot.*

Figure 1 provides a high-level view of how those components are related to each other. In general terms *AnalyserDevice* represents the instrument as a whole. Each *AnalyserDevice* has at least one *AnalyserChannel* and may have *AccessorySlots* through which an *Accessory* can be connected. Similarly, each *AnalyserChannel* may have *AccessorySlots* through which *Accessories* can be connected. Data acquisition occurs through the *AnalyserChannel* or through the *Accessory* connected to that *AnalyserChannel*. *Accessories* can only be connected through the *AccessorySlots*.

The interface with the process to monitor is done through a sampling system that connects the *AnalyserChannel* to a specific *sampling point* in the process. This connection is also referred as a *Stream*.

To decrease the cost of the analyser per *sampling point*, some analysers use sampling systems that can multiplex more than one *sampling point*. These systems are often referred to as multi-stream analysers.

More than one AnalyserChannel can collect data from the process at the same time, but only one Stream may be active at a given time on an AnalyserChannel.

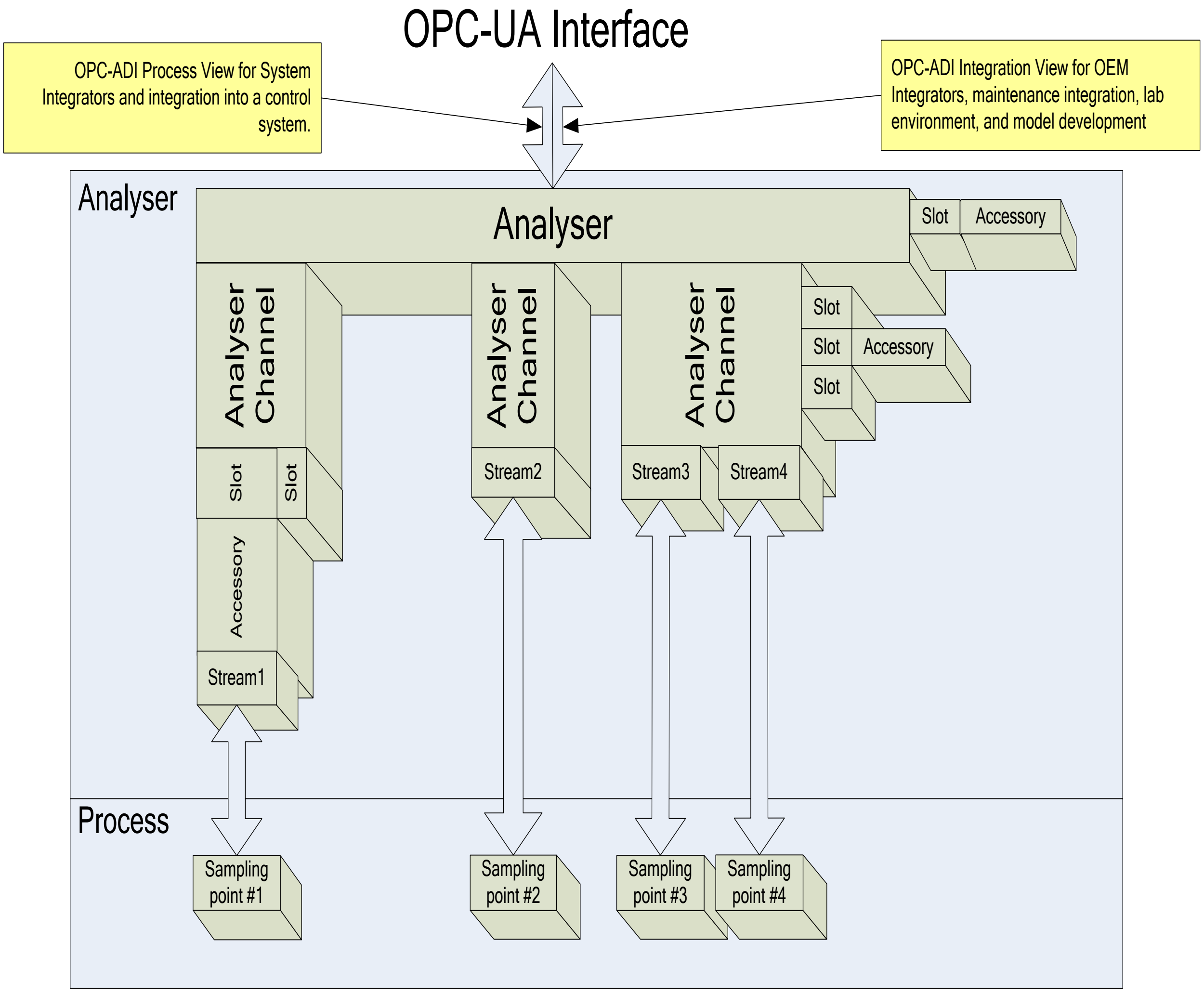

Figure 1 – High Level Object Model overview

For a detailed overview diagram of the ADI object model, refer to Figure 2. Elements illustrated in that diagram are further described in separate sections of this document..

5 Model

The following paragraphs describe the elements of the ADI *Information Model*. All elements of the ADI *Information Model* defined by this specification belong to OPC-ADI namespace unless otherwise specified. OPC-ADI namespace is identified by the following URI:

http://opcfoundation.org/UA/ADI/

5.1 General

Figure 2 illustrates the overview of the ADI object model. It illustrates main components of the object model in the OPC-UA notation as described in Appendix D of [UA Part 3].

AnalyserDeviceType, *AnalyserChannelType*, *StreamType*, *AccessorySlotType* and *AccessoryType* represent the main building blocks of the object model. They are described in detail in dedicated paragraphs of this specification. Object of type *AnalyserDeviceType* is the topmost *Object* of the ADI object model. It represents an abstract type which shall be subtyped for different types of analyser devices. Subtypes of *AnalyserDeviceType* are described in 5.2.1.3.

This specification does not attempt to define all *Parameters* for analysers or their components. Instead, it aims to provide a set of mandatory and optional *Parameters* which are common for all analysers or analysers within the same class (type). Additionally, this specification defines placeholders (*FunctionalGroups)* where instrument vendors can expose their custom *Parameters*.

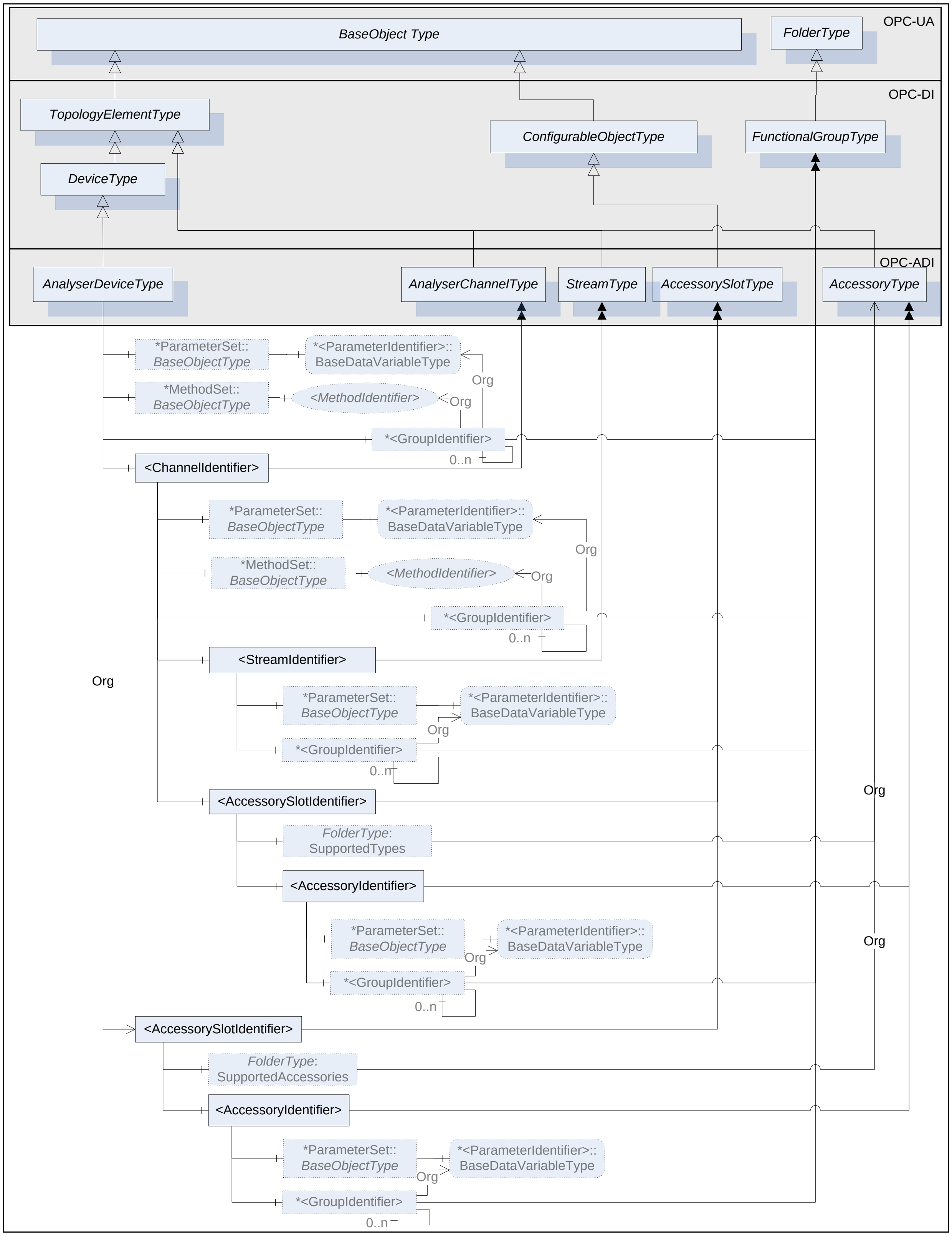

Figure 2 - Object Model Overview

5.2 Object Types

5.2.1 AnalyserDevice

5.2.1.1 Type definition: AnalyserDeviceType *ObjectType*

AnalyserDeviceType defines the general structure of an Analyser*Device Object*. Figure 3, Figure 4 and Figure 5 show the inheritance hierarchy and detailed composition of *AnalyserDeviceType*. It is formally defined in Table 1.

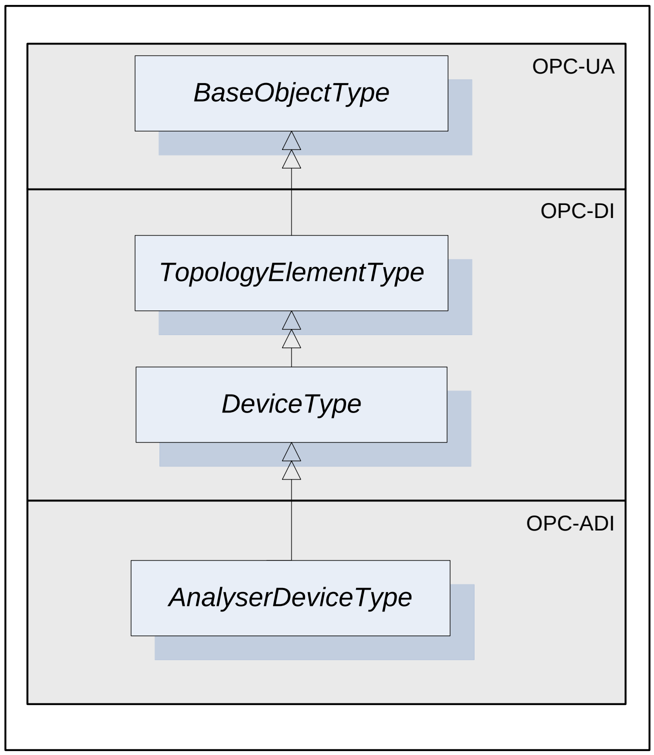

Figure 3 - AnalyserDeviceType

AnalyserDeviceType is a subtype of *DeviceType* [UA-DI] and as such can have *Parameters* which are kept in an *Object* called *ParameterSet*. *Parameters* represented by <*ParameterIdentifier*> and their list called *ParameterSet* are inherited from *DeviceType*.

TopologyElementType [UA-DI] introduced a component called *MethodSet*, which can be used to organize *Methods* exposed to the *Client*. *AnalyserDeviceType* takes advantage of that inherited component and groups all of its *Methods* under *MethodSet*.

DeviceType also introduces *FunctionalGroups* identified by <*GroupIdentifier*> that expose its *Parameters* in an organized fashion reflecting the structure of the device. *AnalyserDeviceType* can have any number of *FunctionalGroups.*

AnalyserDeviceType defines three mandatory *FunctionalGroups:*

- *Configuration* - used to organize *Parameters* representing the high-level configuration items of the analyser, which are expected to be modified by end users.
- *Status* - used to organize *Parameters* which describe the general health of the analyser.
- *FactorySettings* - used to organize *Parameters*, which describe the factory settings of the analyser that are not expected to be modified by end users.

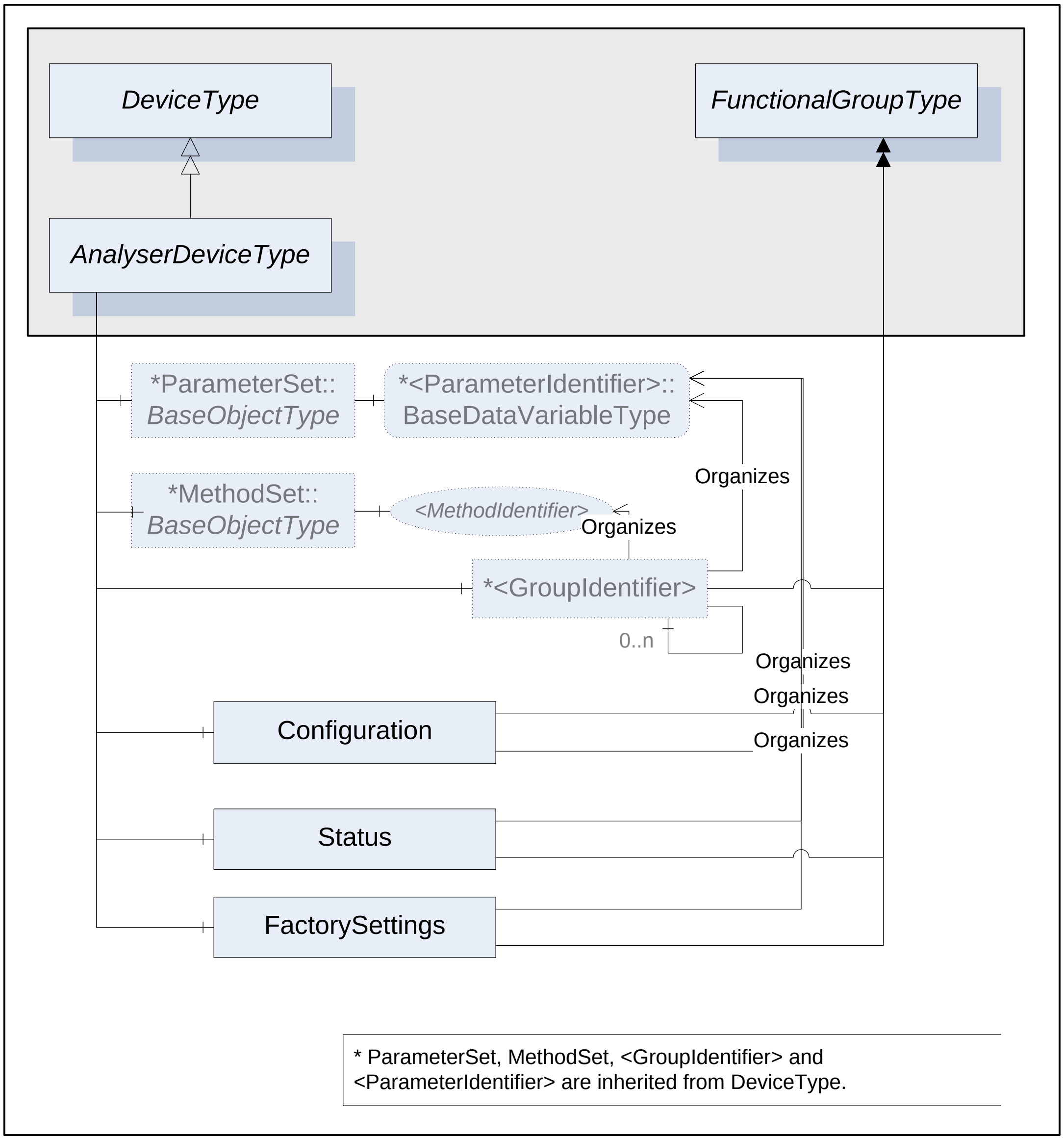

Figure 4 – AnalyserDeviceType Components

The *AnalyserDevice Object* that represents an analyser has one or more *AnalyserChannels*. *AnalyserChannel* is described in clause 5.2.2. The *AnalyserChannel Node* instances are identified by <ChannelIndentifier> browse name.

AnalyserDevice Object has zero or more *Object*s of type *AccessorySlotType* and identified by <*AccessorySlotIdentifier*>. *AccessorySlotType* is described in clause5.2.4. *AccessorySlot Object*s represent physical locations on the analyser where the analytical accessory can be mounted. Accessories currently mounted on the analyser device as well as the supported accessories for the accessory slot are represented as components of the *AccessorySlot Object*. For details refer to clause 5.2.3.

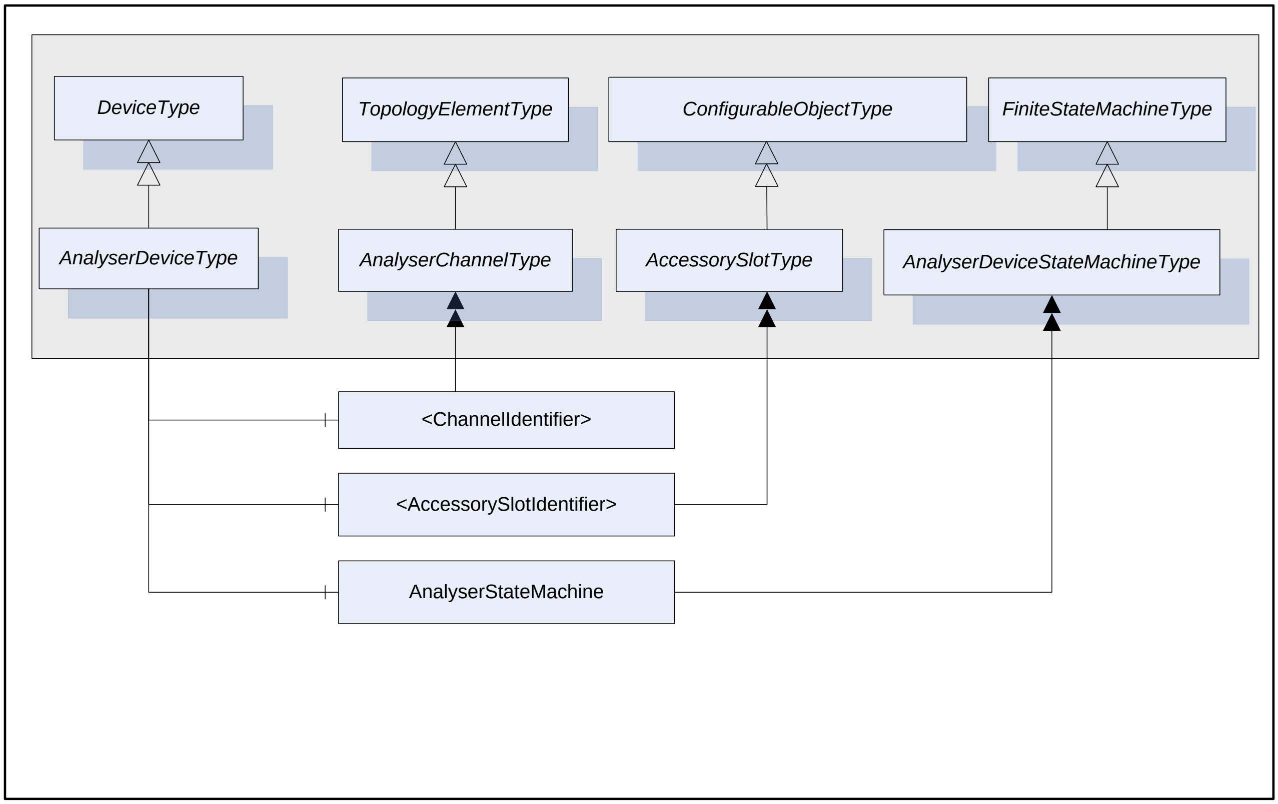

Figure 5 - AnalyserDeviceType Components cont.

AnalyserDeviceType does not expose any mandatory *Parameters* to report or manipulate the state of an analyser device. Instead, *AnalyserDevice* states are exposed through the *AnalyserStateMachine* component of type *AnalyserDeviceStateMachineType*. For details on *AnalyserDeviceStateMachineType* see clause 5.3.2.

Table 1 - AnalyserDeviceType Definition

Attribute	Value				
BrowseName	AnalyserDeviceType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *DeviceType* defined in [UA-DI]					
HasSubtype	ObjectType	SpectrometerDeviceType	Defined in Clause 5.2.6.1		
HasSubtype	ObjectType	ParticleSizeMonitorDeviceType	Defined in Clause 5.2.8.1		
HasSubtype	ObjectType	AcousticSpectrometerDeviceType	Defined in Clause 5.2.9.1		
HasSubtype	ObjectType	MassSpectrometerDeviceType	Defined in Clause 5.2.7.1		
HasSubtype	ObjectType	ChromatographDeviceType	Defined in Clause 5.2.10.1		
HasSubtype	ObjectType	NMRDeviceType	Defined in Clause 5.2.11.1		
HasComponent	Object	Configuration		FunctionalGroupType	Mandatory
HasComponent	Object	Status		FunctionalGroupType	Mandatory
HasComponent	Object	FactorySettings		FunctionalGroupType	Mandatory
HasComponent	Object	<ChannelIdentifier>		AnalyserChannelType	OptionalPlaceHolder
HasComponent	Object	<AccessorySlotIdentifier>		AccessorySlotType	OptionalPlaceHolder
HasComponent	Object	AnalyserStateMachine		AnalyserDeviceStateMachineType	Mandatory
AnalyserDeviceType.MethodSet					
HasComponent	Method	GetConfiguration			Mandatory
HasComponent	Method	SetConfiguration			Mandatory
HasComponent	Method	GetConfigDataDigest			Mandatory
HasComponent	Method	CompareConfigDataDigest			Mandatory
HasComponent	Method	ResetAllChannels			Mandatory
HasComponent	Method	StartAllChannels			Mandatory
HasComponent	Method	StopAllChannels			Mandatory
HasComponent	Method	AbortAllChannels			Mandatory
HasComponent	Method	GotoOperating			Mandatory
HasComponent	Method	GotoMaintenance			Mandatory

AnalyserDeviceType is a subtype of *DeviceType* defined in [UA-DI] and as such it inherits *DeviceType*'s characteristics. For a complete definition of the *DeviceType* see [UA-DI].

5.2.1.2 AnalyserDevice *Object*

The *AnalyserDeviceType ObjectType* is abstract. There will be no instances of an *AnalyserDeviceType* itself, but there will be instances of sub-types of this type. In this specification, the term *AnalyserDevice* generically refers to an instance of any *ObjectType* derived from the *AnalyserDeviceType ObjectType*.

All *AnalyserDevices* have *Attributes* and *Properties* that they inherit from the *DeviceType*. For those *elements*, the same rules as defined for *Device Object*s in [UA-DI] apply.

5.2.1.3 Sub-types of AnalyserDeviceType *ObjectType*

The sub types of the AnalyserDeviceType are illustrated in Figure 6. Each of these sub type may be further sub typed.

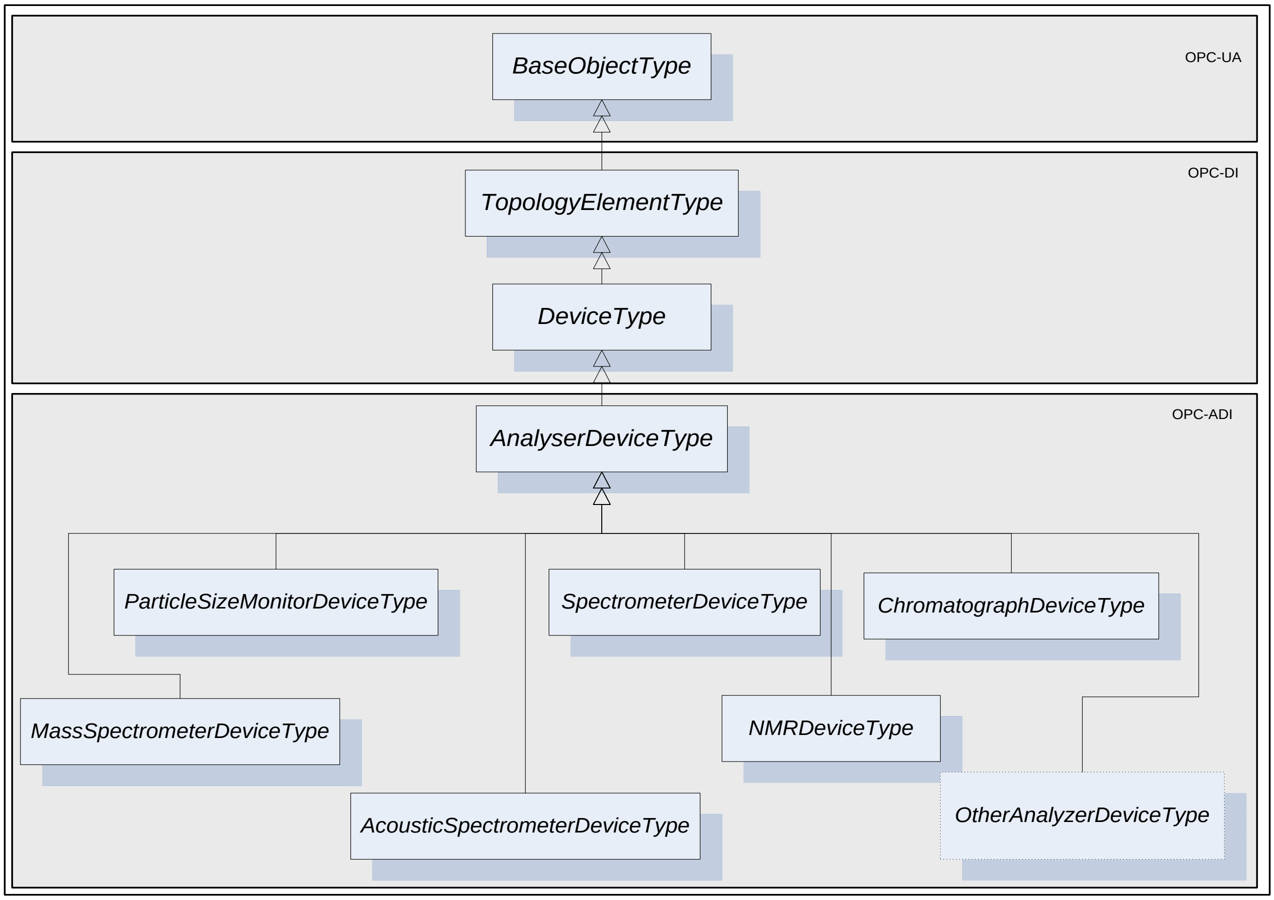

Figure 6 - AnalyserDeviceType Hierarchy

The *AnalyserDeviceType* is derived from the *DeviceType* as an Abstract type. It is sub-typed for each one of the analyser classes. Six sub-types are introduced:

Table 2 –AnalyserDeviceType Sub-type definition

AnalyserDeviceType	**Description**
SpectrometerDeviceType	A light spectrometer is an optical instrument used to measure Properties of light over a specific portion of the electromagnetic spectrum (IR/NIR/VIS/UV), typically used in spectroscopic analysis to identify chemical composition of sample materials. The use of analytical techniques to determine process control parameters from spectra allows a wide range of industrial applications. This type covers FTIR, diode array, etc.
AcousticSpectrometerDeviceType	An acoustic spectrometer uses sound wave emission and advanced pattern recognition software to predict the physical Properties of powders and particulates. This type of analyser uses high frequency sounds emitted by all physical and chemical processes (particle impact, turbulent gas flow, gas evolution, fermentation, cavitation and multiphase flow). It is a non-invasive technique which is responding to dynamic event making it suitable for process control.
MassSpectrometerDeviceType	A mass spectrometer is an analytical instrument used to measure the mass-to-charge ratio of ions. It is most generally used to find the composition of a physical sample by generating a mass spectrum representing the masses of sample components. A wide range of industrial process control applications are therefore possible, such as the online control of solvent drying.

AnalyserDeviceType	Description
ParticleSizeMonitorDeviceType	Particle size can be determined by light scattering (e.g. Focus Beam Reflectance Measurement) or other *Method*s. This type of analyser can be used to implement particle monitoring technique for in-line real-time measurement of particle size. A wide range of industrial process control applications are therefore possible such as the online control of crystallizers
ChromatographDeviceType	Chromatography is the collective term for a family of techniques for the separation of mixtures. It involves passing a mixture dissolved in a "mobile phase" through a stationary phase, which separates the analyte to be measured from other molecules in the mixture and allows it to be isolated. Chromatography may be preparative or analytical. Preparative chromatography seeks to separate the components of a mixture for further use (and is thus a form of purification). Analytical chromatography normally operates with smaller amounts of material and seeks to measure the relative proportions of analytes in a mixture. The two are not mutually exclusive
NMRDeviceType	Nuclear Magnetic Resonance spectrometers

5.2.1.4 Parameters of AnalyserDeviceType

Parameters defined for the *AnalyserDeviceType* are described in the following tables. The tables correspond to mandatory *FunctionalGroups* defined for the *AnalyserDeviceType*. Additional *Parameters* may be defined on subtypes of *AnalyserDeviceType* and associated with those *FunctionalGroups*.

All *AnalyserDevice Parameters* exist as components of *ParameterSet Object* defined on that *AnalyserDevice* through inheritance from *DeviceType*. Each *Parameter* defined for an *AnalyserDevice* shall be accessible through one or more *FunctionalGroup* defined on that *AnalyserDevice*. Note, that the same *Parameter* is not instantiated more than once. Both, *ParameterSet* and a specific *FunctionalGroup* maintain *Reference*s to the same instance of the *Parameter*.

Table 3 shows *Parameters* that will be organized by the *Configuration FunctionalGroup*.

Table 3 – AnalyserDevice Configuration Parameters

BrowseName	Description	VariableType	Optional/ Mandatory
ConfigData	Optional representation of the AnalyserDevice configuration	FileType	O

ConfigData is an optional representation of the AnalyserDevice configuration. When it is present, it may be used to read and write the AnalyserDevice configuration in chunks. The main purpose of this element is to provide a way to read and write configuration that are larger than the maximum size of the OPC UA message. Reading and writing configuration through this object are subject to the same state machine constraints as GetConfiguration and SetConfiguration.

To maintain configuration consistency, the server must grant read and write access to one and only one user at any given time.

The steps to update the configuration through the ConfigData object are:

1. When SetConfiguration is allowed based on the state machine states, a single user may cal “open” the ConfigData. If an “Open” is attempted when not permitted, the server shall return “Bad_InvalidState”.

2. The user updates the configuration by calling repeatitively and in increasing order “write” method on ConfigData. If the “Write” are not sequential, the server shall return “Bad_InvalidArgument”.

3. When the whole configuration has been written, the user calls “close” method on the ConfigData.

4. The server is responsible to verify the configuration. If an error occurs during the verification, the server shall return “Bad_InvalidArgument” on the “Close”. In case of error, the previous configuration is restored.

5. The server commits the new configuration. . If an error occurs during the commit, the server shall return “Bad_InvalidArgument” on the “Close”. In case of error, the previous configuration is restored.

Table 4 shows *Parameters* that will be organized by the *Status FunctionalGroup*. All *Parameters* organized by this *FunctionalGroup* shall be read-only.

Table 4 – AnalyserDevice Status Parameters

BrowseName	**Description**	**VariableType**	**Optional/ Mandatory**
DiagnosticStatus	General health status of the analyser	DataItemType DataType=DeviceHealthEnumeration	M

The *DiagnosticStatus Parameter* reflects the general health of analyser. It is defined as a *Variable* of *DataItemType* type and its possible values are defined by [UA-DI] enumeration*DeviceHealthEnumeration*. Its value must be the same as DeviceType.DeviceHealth Property.

Table 5 shows *Parameters* that will be organized by the *FactorySettings FunctionalGroup* component of the *AnalyserDeviceType*.

Table 5 – AnalyserDevice FactorySettings Parameters

BrowseName	**Description**	**VariableType**	**Optional/ Mandatory**

The SerialNumber, Manufacturer, Model, DeviceManual, DeviceRevision, SoftwareRevision and the HardwareRevision *Properties* are defined on *DeviceType* and as such available on AnalyserDeviceType. As a general rule, they are read-only *properties*. However, they can be updated to reflect changes made to the analyser configuration e.g. upgrading the firmware.

DeviceRevision *Property* will be used to indicate an overall change in the analyser. It is mandatory and shall be updated automatically or manually each time the analyser configuration is altered. It is the customer’s QA responsibility to determine if this particular change affects the validation of the analyser.

The RevisionCounter *Property* is an incremental counter indicating the number of times the semi-static data within the *AnalyserDevice* has been modified.

If the analytical device represented by an *AnalyserDevice Object* is unable to publish a value for a mandatory *Parameter* defined in Table 5, the Analyser Server should provide a way to manually enter that value.

5.2.1.5 Methods of AnalyserDeviceType

All *Methods* defined for *AnalyserDeviceType* and its state machines are grouped under the *MethodSet* component inherited from *DeviceType* [UA-DI].

AnalyserDeviceType defines a *Method* called *GetConfiguration*, which is used to read the complete configuration of the *AnalyserDevice* and all of its components (*AnalyserChannel*, *Accessory*, *AccessorySlot etc.)* from the Analyser Server*.* The configuration is a proprietary structure defined by the analyser vendor, and is represented as a ByteString.

AnalyserDeviceType defines a *Method* called *SetConfiguration*, which is used to write the complete configuration of the *AnalyserDevice* and all of its components to the Analyser Server. This *Method* can be executed only when all of the *AnalyserChannels* are in a Stopped state or in a Maintenance state (see 5.3.4.3). An attempt to call it while in any other state results in a failure of the *Method* call.

When the *SetConfiguration Method* is executed, it automatically causes a transition of all *AnalyserChannels* in a Stopped state to the Resetting state and the new configuration becomes active. The configuration is a structure provided by the analyser vendor, and represented as a ByteString.

Even if the ADI *Client* verifies the configuration before calling the *SetConfiguration Method*, the Analyser Server has the ultimate responsibility to verify the configuration (*Parameter* ranges, *Parameter* values relating to each other, *Parameter* values in regard to installed hardware) before applying the requested changes. If any *Parameter* value is invalid, the whole configuration shall be rejected.

If an error occurs during a method call, the analyser state should be returned the same as before the call or at least a stable state.

Table 6 – GetConfiguration Method

Method	**Description**			
GetConfiguration	Read the complete configuration of the AnalyserDevice and all of its components to the Analyser *Server*.			
	InputArguments			
	Name	**DataType**	**ValueRank / arrayDimension**	**Description**
		N/A	N/A	
	OutputArguments			
	Name	***DataType***	***arraySize/ arrayDimension***	***Description***
	ConfigData	ByteString	-1/[0]	Configuration structure represented as a single dimensional array of Bytes. Length of an array is provided by the *Server* at runtime. If the size of ConfigData parameter is larger than a single OPC UA message, the AnalyserDevice.ConfigData object shall be used.

Table 7 – SetConfiguration Method

Method	**Description**			
SetConfiguration	Write the complete configuration of the AnalyserDevice and all of its components to the Analyser Server and make the new configuration active.			
	InputArguments			
	Name	**DataType**	**ValueRank / arrayDimension**	**Description**
	ConfigData	ByteString	-1/[0]	Configuration structure represented as a single dimensional array of Bytes. Length of an array is provided by the *Client* at runtime.

				If the size of ConfigData parameter is larger than a single OPC UA message, the AnalyserDevice.ConfigData object shall be used.
	OutputArguments			
	Name	***DataType***	***arraySize/ arrayDimension***	***Description***
	ConfigDataDigest	String	-1/[0]	Vendor specific digest (like SHA1) of the ConfigData. It is calculated, by the *Server*, after ConfigData is received and before any change has been made. It is used as the reference to know if the configuration has been altered after the SetConfiguration call. This string is intended to be human readable for example the hexadecimal or Base64 representation of the SHA1.

AnalyserDevice defines a *Method* called *GetConfigDataDigest*, which is used to read the digest (e.g. SHA1 hash) of the complete analyser configuration. The digest is returned in a *Method* argument called *ConfigDataDigest*. It represents the same data which is calculated by the *Server*, when *SetConfiguration Method* is called. The value returned in *ConfigDataDigest* will change when the configuration of the analyser is changed in a way that may alter the results it produces. Examples of analyser changes that may affect the value of *ConfigDataDigest* are:

a) A configuration *Parameter* of the analyser or any of its components is modified. There are rare cases where a change of a *Parameter* does not affect the analyser results like setting an acquisition trigger. In these cases the *ConfigDataDigest* shall not be recomputed. The vendor shall clearly specify which *Parameters* do not affect *ConfigDataDigest*.

b) A *Method* call which does not update *Parameters* but alters behaviour of the analyser (e.g. firmware update) is called. The vendor shall clearly specify which *Method*s affect the returned value from *ConfigDataDigest*

c) An accessory is added or removed

d) Analyser is configured locally via built-in panel.

By comparing the *ConfigDataDigest* output argument from the *SetConfiguration Method* with the current value returned in the *ConfigDataDigest* argument of the *GetConfigDataDigest Method*, a *Client* shall be able to determine if the analyser configuration has been modified in such a way that the results produced by the analyser may be different than expected.

Table 8 – GetConfigDataDigest Method

Method	**Description**			
GetConfigDataDigest	Read the digest of the complete analyser configuration as computed by the *Server*.			
	InputArguments			
	Name	**DataType**	**ValueRank / arrayDimension**	**Description**
	None	N/A	N/A	
	OutputArguments			
	Name	***DataType***	***arraySize/ arrayDimension***	***Description***
	ConfigDataDigest	String	-1/[0]	Vendor specific digest (like SHA1) of the complete analyser configuration. It

				is used as the reference to know if the configuration has been altered after the last SetConfiguration call. This string is intended to be human readable for example the hexadecimal or Base64 representation of the SHA1.

A *Method* called *CompareConfigDataDigest* can be used to ask the AnalyserDevice if the *ConfigDataDigest* held by the *Client* reflects the current configuration of the analyser. This approach relieves the client from the responsibility for comparing the configuration digests.

Table 9 – CompareConfigDataDigest Method

Method	**Description**			
CompareConfigDataDigest	Compare the provided ConfigDataDigest with the actual one of the analyser.			
	InputArguments			
	Name	**DataType**	**ValueRank / arrayDimension**	**Description**
	ConfigDataDigest	String	-1/[0]	Vendor specific digest (like SHA1) of the complete analyser configuration as returned by SetConfiguration and GetConfigurationDataDigest. This string is intended to be human readable for example the hexadecimal or Base64 representation of the SHA1.
	OutputArguments			
	Name	***DataType***	***arraySize/ arrayDimension***	***Description***
	IsEqual	Boolean	-1/[0]	True if the input ConfigDataDigest is equal to the actual digest of the analyser configuration.

AnalyserDeviceType defines several *Methods* used for simultaneous control of analyser channels. Those *Methods* are defined in the following tables.

Table 10 – ResetAllChannels Method

Method	**Description**
ResetAllChannels	Reset all AnalyserChannels belonging to this AnalyserDevice.
	InputArguments: NONE
	OutputArguments: NONE

Table 11 – StartAllChannels Method

Method	**Description**
StartAllChannels	Start all AnalyserChannels belonging to this AnalyserDevice.
	InputArguments: NONE

	OutputArguments: NONE

Table 12 – StopAllChannels Method

Method	**Description**
StopAllChannels	Stop all AnalyserChannels belonging to this AnalyserDevice.
	InputArguments: NONE
	OutputArguments: NONE

Table 13 – AbortAllChannels Method

Method	**Description**
AbortAllChannels	Abort all AnalyserChannels belonging to this AnalyserDevice.
	InputArguments: NONE
	OutputArguments: NONE

Methods described in Table 10, Table 11, Table 12, Table 13 operate on all *AnalyserChannels* that are in the *Operating* state and their *Configuration.IsEnabled Parameter* is set to True. These *Method*s are not guaranteed to be atomic and their effect on each *AnalyserChannel* is not necessarily simultaneous. For example, the following implementation is perfectly legal:

```
For each AnalyserChannel
    If AnalyserChannel.IsInOperatingState AND
            AnalyserChannel.Configuration.IsEnabled == TRUE
        AnalyserChannel.Reset ()
```

Table 14 - GotoOperating Method

Method	**Description**
GotoOperating	Causes the AnalyserDeviceStateMachine to go to Operating state, forcing all AnalyserChannels to leave the SlaveMode state and go to the Operating state.
	InputArguments: NONE
	OutputArguments: NONE

Table 15 - GotoMaintenance Method

Method	**Description**
GotoMaintenance	Causes the AnalyserDeviceStateMachine to go to Maintenance state, forcing all AnalyserChannels to SlaveMode state..

	InputArguments: NONE
	OutputArguments: NONE

Table 16 – Method result codes for AnalyserDeviceType methods

Result code	Description
Bad_InvalidArgument	One or more argument re invalid.
Bad_InvalidState	Method called w hen the analyser is not in the appropriate state.
Bad_RequestTooLarge	The request message size exceeds limits set by the server.
Bad_ResponseTooLarge	The response message size exceeds limits set by the client.
Bad_ServiceUnsupported	The analyser does not support the requested service.
Bad_UnexpectedError	An unexpected error occurred.

5.2.2 AnalyserChannel

5.2.2.1 Type definition: AnalyserChannelType *ObjectType*

This *ObjectType* defines the structure of an *AnalyserChannel Object*. Figure 7 depicts the *AnalyserChannelType* hierarchy. Figure 8 and Figure 9 show the *AnalyserChannelType* components. It is formally defined in Table 17.

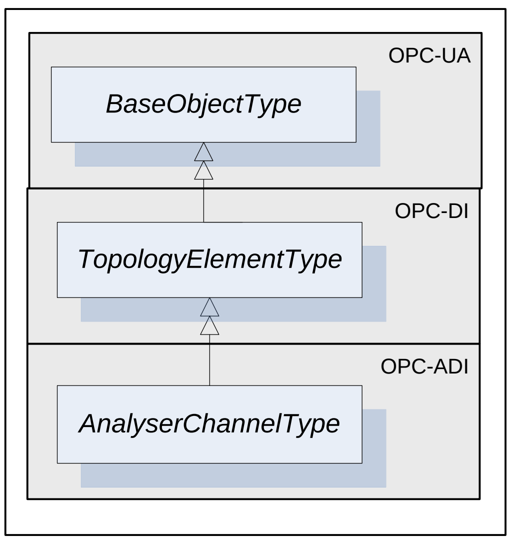

Figure 7 - AnalyserChannelType

AnalyserChannelType is a subtype of *TopologyElementType*.

An *AnalyserChannel* may have *Parameters*. If an *AnalyserChannel* has *Parameters* they appear in an *Object* called *ParameterSet* as a flat list of *Parameters*. *ParameterSet* is inherited from *TopologyElementType* [UA-DI]. *Parameters* of an *AnalyserChannel* are identified by the <ParameterIdentifier> browse name.

TopologyElementType [UA-DI] introduces a component called *MethodSet*, which shall be used to organize *Methods* exposed to the *Client*. *AnalyserChannelType* takes advantage of that inherited component and groups all of its *Methods* and the ones from its substate machines under *MethodSet*.

Parameters of an *AnalyserChannel* can be organized in *FunctionalGroups* identified as <GroupIdentifier> browse name.

AnalyserChannelType defines two mandatory *FunctionalGroups (see clause 5.2.1.4 for details)*:

- *Configuration* - used to organize *Parameters* representing the high-level configuration items of the channel, which are expected to be modified by end users.
- *Status* - used to organize *Parameters* which describe the general health of the channel.

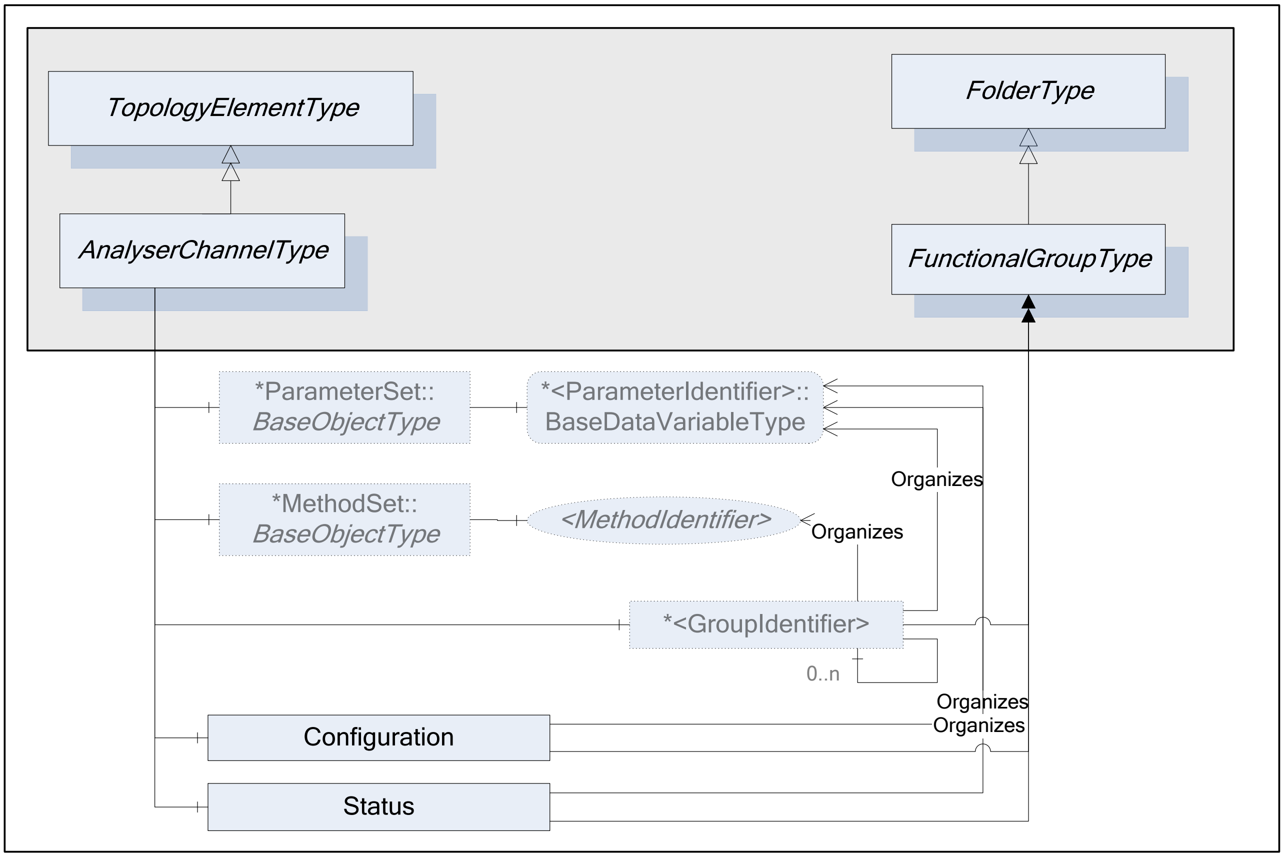

Figure 8 - AnalyserChannelType FunctionalGroups

AnalyserChannel Object has zero or more *Objects* of type *AccessorySlotType* and identified by *<AccessorySlotIdentifier> browse name*. *AccessorySlotType* is described in clause 5.2.3. *AccessorySlot Objects* represent physical locations on the physical channel where the analytical accessory can be mounted. Accessories currently mounted on the analyser channel as well as the supported accessories for the *AccessorySlot* are defined as components of the *AccessorySlot Object*. For details refer to clause 5.2.3.

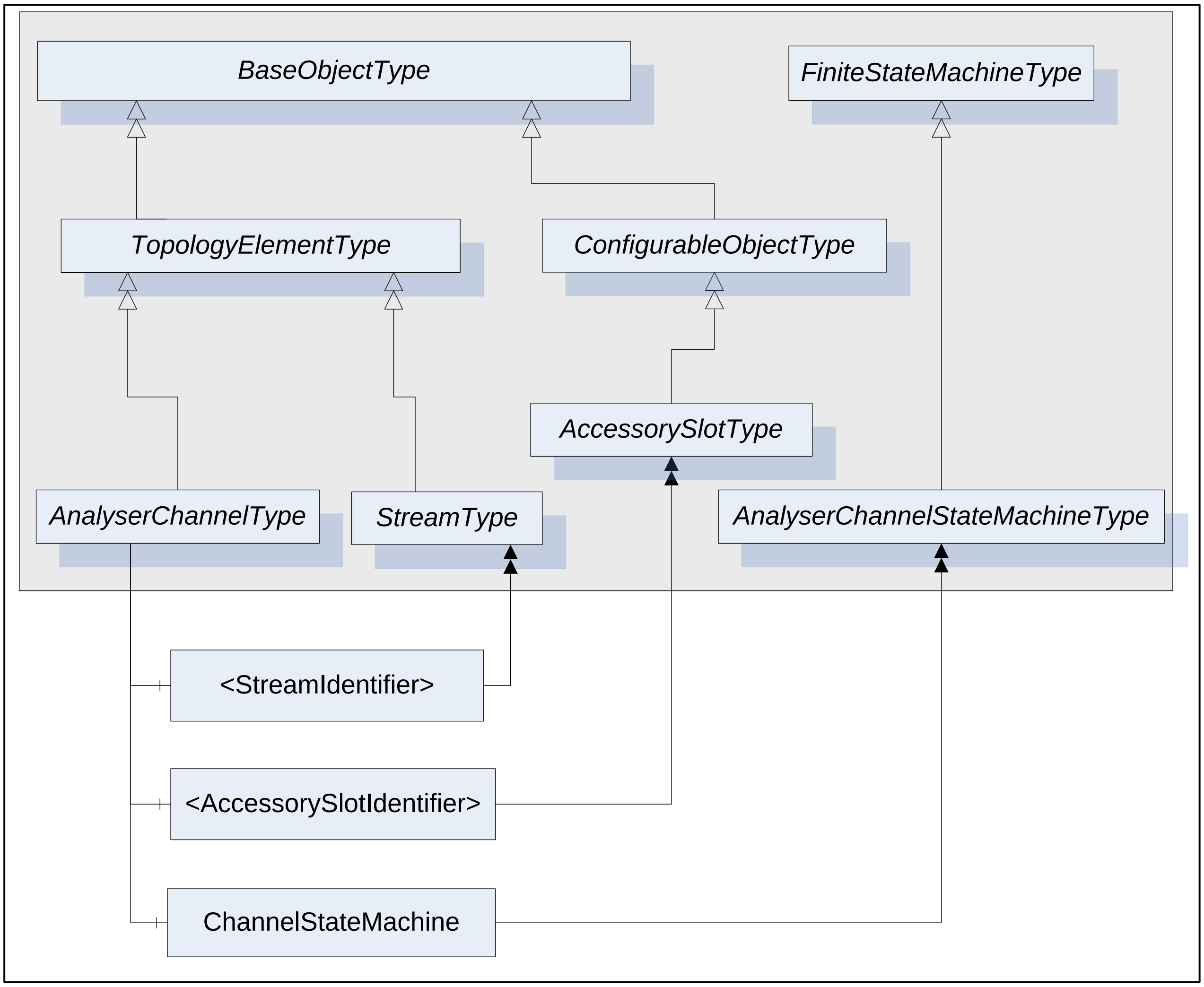

Figure 9 - AnalyserChannelType Components

AnalyserChannelType does not expose any mandatory *Parameters* to report or manipulate the state of an *AnalyserChannel*. Instead, *AnalyserChannel* states are exposed through the *ChannelStateMachine Object* of the type *AnalyserChannelStateMachineType*. For details on *AnalyserChannelStateMachineType* see clause 5.3.2.

Table 17 – AnalyserChannelType Definition

Attribute	Value				
BrowseName	AnalyserChannelType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *TopologyElementType* defined in [UA-DI].					
HasComponent	Object	ParameterSet		BaseObjectType	Mandatory
HasComponent	Object	<GroupIdentifier>		FunctionalGroupType	OptionalPlaceHolder
HasComponent	Object	Configuration		FunctionalGroupType	Mandatory
HasComponent	Object	Status		FunctionalGroupType	Mandatory
HasComponent	Object	<StreamIdentifier>		StreamType	OptionalPlaceHolder
HasComponent	Object	<AccessorySlotIdentifier>		AccessorySlotType	OptionalPlaceHolder
HasComponent	Object	ChannelStateMachine		AnalyserChannelStateMachineType	Mandatory
AnalyserChannelType.MethodSet					
HasComponent	Method	GotoOperating			Mandatory
HasComponent	Method	GotoMaintenance			Mandatory
HasComponent	Method	StartSingleAcquisition			Mandatory
HasComponent	Method	Reset			Mandatory
HasComponent	Method	Start			Mandatory
HasComponent	Method	Stop			Mandatory
HasComponent	Method	Hold			Mandatory
HasComponent	Method	Unhold			Mandatory
HasComponent	Method	Suspend			Mandatory
HasComponent	Method	Unsuspend			Mandatory
HasComponent	Method	Abort			Mandatory
HasComponent	Method	Clear			Mandatory

5.2.2.2 AnalyserChannel *Object*

The term *AnalyserChannel* refers to an instance of the *AnalyserChannelType ObjectType* as defined in Table 17.

All *AnalyserChannels* have *Attributes* and *Properties* inherited from the BaseObject.

Each *AnalyserDevice Object* has at least one *AnalyserChannel Object* as its component.

5.2.2.3 Parameters of AnalyserChannelType

Parameters defined for the *AnalyserChannelType* are described in the following tables. The tables correspond to mandatory *FunctionalGroups* defined for the *AnalyserChannelType*. Additional *Parameters* may be defined for *AnalyserChannel* on subtypes of *AnalyserDeviceType* and associated with those *FunctionalGroups*.

All *AnalyserChannel Parameters* exist as components of the *ParameterSet Object* defined on that *AnalyserChannel*. Each *Parameter* defined for an *AnalyserChannel* shall be accessible through one and only one *FunctionalGroup* defined on that *AnalyserChannel*. Note, that the same *Parameter* is not instantiated more than once. Both, *ParameterSet* and a specific *FunctionalGroup* maintain *References* to the same instance of the *Parameter*.

Table 18 shows *Parameters* that will be organized by the *Configuration FunctionalGroup*.

Table 18 – *AnalyserChannel* Configuration Parameters

BrowseName	Description	VariableType	Optional/ Mandatory
ChannelId	Channel Id defined by user. On some analysers, the name of a channel may be configured using a maintenance tool, which leads to having two names to refer to the same channel for example: Channel1 and FirstChannel. In this case, one is for the BrowseName and the second is the ChannelId.	DataItemType (DataType=String)	O
IsEnabled	True if this AnalyserChannel maybe used to perform acquisition. Allow an AnalyserChannel to be marked as “not in use” so xxxAllChannels *Method*s of the AnalyserDevice may skip it. In the case of “software” AnalyserChannel like GC, this allows a chromatographic application to be disabled.	DataItemType (DataType=Boolean)	M

Table 19 shows *Parameters* that will be organized by *Status FunctionalGroup*. All *Parameters* organized by this *FunctionalGroup* shall be read-only.

Table 19 – *AnalyserChannel* Status Parameters

BrowseName	Description	VariableType	Optional/ Mandatory
DiagnosticStatus	AnalyserChannel health status	*DataItemType* (DataType=DeviceHealthEnumeration)	M
ActiveStream	Active stream for this AnalyserChannel. Its value is the BrowseName of the active stream. If no Stream is active, it shall be set to NULL.	DataItemType (DataType=String)	M

The *DiagnosticStatus Parameter* reflects the general health of the channel. It is defined as a *Variable* of *DataItemType* type and its value is defined by [UA-DI] enumeration *DeviceHealthEnumeration*.

5.2.2.4 Methods of AnalyserChannelType

All *Methods* defined for *AnalyserChannelType* and its substate machines are grouped under the *MethodSet* component inherited from *TopologyElementType* [UA-DI].

AnalyserChannel defines a *Method* called *StartSingleAcquisition*, which is used to start a single data acquisition, which uses current values of *Parameters* from the *AcquisitionSettings FunctionalGroup* of the Stream indicated by *SelectedStream* argument. The *Method* argument *ExecutionCycle* is used to indicate what it is that the acquisition is collecting e.g. sample, background, and dark noise.

If an error occurs during a method call, the analyser state should be the same as before the call.

Table 20 – *StartSingleAcquisition Method*

Method	Description
StartSingleAcquisition	Start collection of a single sample or reference data
	InputArguments

	Name	DataType	ValueRank / arrayDimension	Description
	ExecutionCycle	ExecutionCycleEnumeration	-1/[0]	Enumeration which specifies the type of the acquisition cycle (e.g. Calibration, Sampling)
	ExecutionCycleSubcode	UInteger	-1/[0]	Vendor defined code, which further describes the acquisition cycle. This code should correspond to one of the enumeration codes defined for *ExecutionCycleSubcode Parameter* in the *AcquisitionStatus FunctionalGroup* on a *Stream*.
	SelectedStream	String	-1/[0]	Browse name of the target Stream for this acquisition
	OutputArguments: NONE			

Table 21 - GotoOperating Method

Method	Description
GotoOperating	Causes the AnalyserChannelStateMachine to go to Operating state..
	InputArguments: NONE
	OutputArguments: NONE

Table 22 - GotoMaintenance Method

Method	Description
GotoMaintenance	Causes the AnalyserChannelStateMachine to go to Maintenance state.
	InputArguments: NONE
	OutputArguments: NONE

Table 23 - Reset Method

Method	Description
Reset	Causes transition to the Resetting state.
	InputArguments: NONE
	OutputArguments: NONE

Table 24 - Start Method

Method	Description
Start	Causes transition to the Starting state.

	InputArguments: NONE
	OutputArguments: NONE

Table 25 - Stop Method

Method	**Description**
Stop	Causes transition to the Stopping state.
	InputArguments: NONE
	OutputArguments: NONE

Table 26 - Hold Method

Method	**Description**
Hold	Causes transition to the Holding state.
	InputArguments: NONE
	OutputArguments: NONE

Table 27 - Unhold Method

Method	**Description**
Unhold	Causes transition to the Unholding state.
	InputArguments: NONE
	OutputArguments: NONE

Table 28 - Suspend Method

Method	**Description**
Suspend	Causes transition to the Suspending state.
	InputArguments: NONE
	OutputArguments: NONE

Table 29 - Unsuspend Method

Method	**Description**
Unsuspend	Causes transition to the Unsuspending state.
	InputArguments: NONE
	OutputArguments: NONE

Table 30 - Abort Method

Method	Description
Abort	Causes transition to the Aborting state.
	InputArguments: NONE
	OutputArguments: NONE

Table 31 - Clear Method

Method	Description
Clear	Causes transition to the Clearing state.
	InputArguments: NONE
	OutputArguments: NONE

Table 32 - Method result codes for AnalyserChannelType methods

Result code	Description
Bad_InvalidArgument	One or more argument re invalid.
Bad_InvalidState	Method called w hen the analyser is not in the appropriate state on one of its state machines.
Bad_RequestTooLarge	The request message size exceeds limits set by the analyser; the ConfigData is too big.
Bad_ResponseTooLarge	The response message size exceeds limits set by the client; the ConfigData is too big.
Bad_ServiceUnsupported	The analyser does not support the requested service.
Bad_UnexpectedError	An unexpected error occurred.

5.2.3 Stream

5.2.3.1 Type definition: StreamType *ObjectType*

This ObjectType defines the structure of a Stream Object. Figure 10 depicts the StreamType hierarchy. It is formally defined in Table 33.

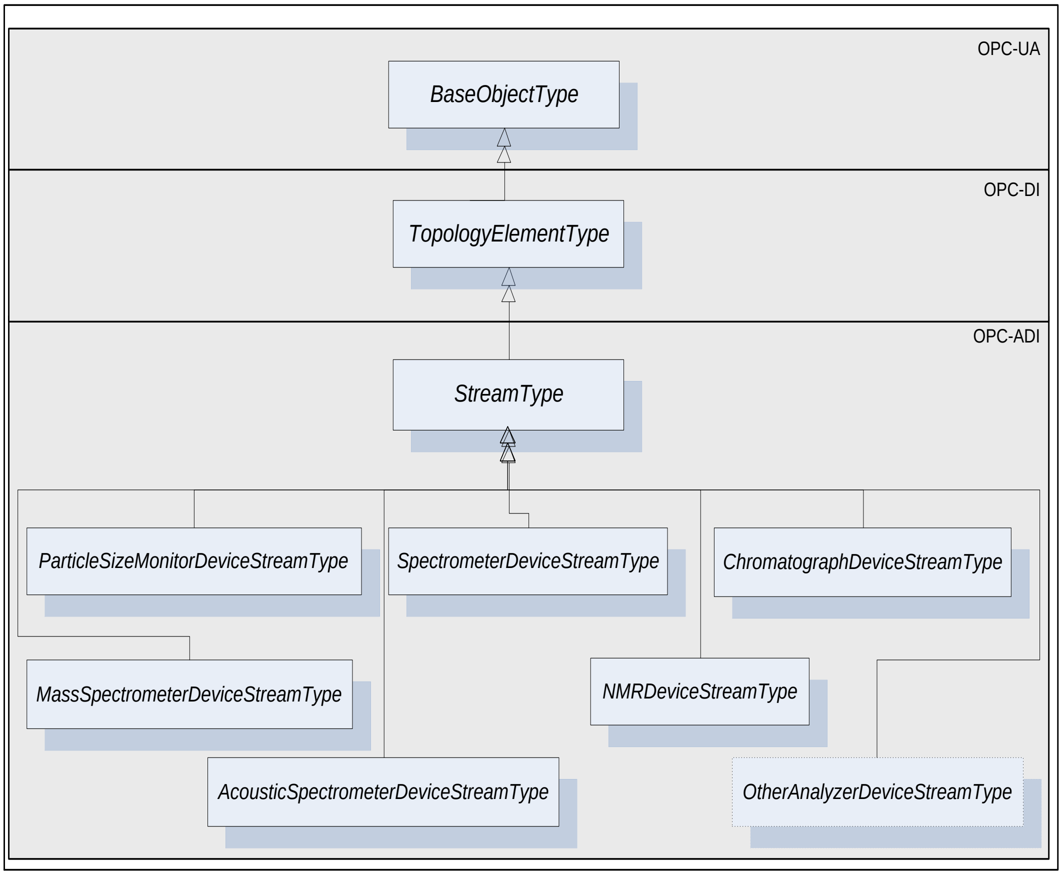

Figure 10 - StreamType Hierarchy

StreamType is a subtype of *TopologyElementType*.

A *Stream* may have *Parameters*. If a *Stream* has *Parameters* they appear in an *Object* called *ParameterSet* as a flat list of *Parameters*. *Parameters* of a *Stream* are identified by the <ParameterIdentifier> browse name. *Parameters* of a *Stream* can be organized in *FunctionalGroups* identified as <GroupIdentifier> browse name.

StreamType defines seven mandatory *FunctionalGroups (see clause 5.2.1.4 for more details)*:

- *Configuration* - used to organize *Parameters* representing the high-level configuration items of the stream, which are expected to be modified by end users.
- *Status* - used to organize *Parameters* which describe the general health of the stream.
- *AcquistionSettings* - used to organize *Parameters* which describe the conditions of the following acquisition on a stream.
- *AcquisitionStatus* – used to organize *Parameters* which describe the status of an ongoing acquisition on a stream.
- *AcquisitionData* - used to organize all *Parameters* which represent data retrieved at the end of the data acquisition.

- *ChemometricModelSettings* - used to organize *Parameters* which describe/configure the chemometric models used during the data acquisition

- *Context* - used to organize all *Parameters* which provide the context for the data acquired through the Stream. Context *Parameters* are not generally used by the analyser but can be published to uniquely tie acquired data with the controlling process. Examples of context *Parameters* are: CampaignID, BatchID, LotID, MaterialID, and SampleId.

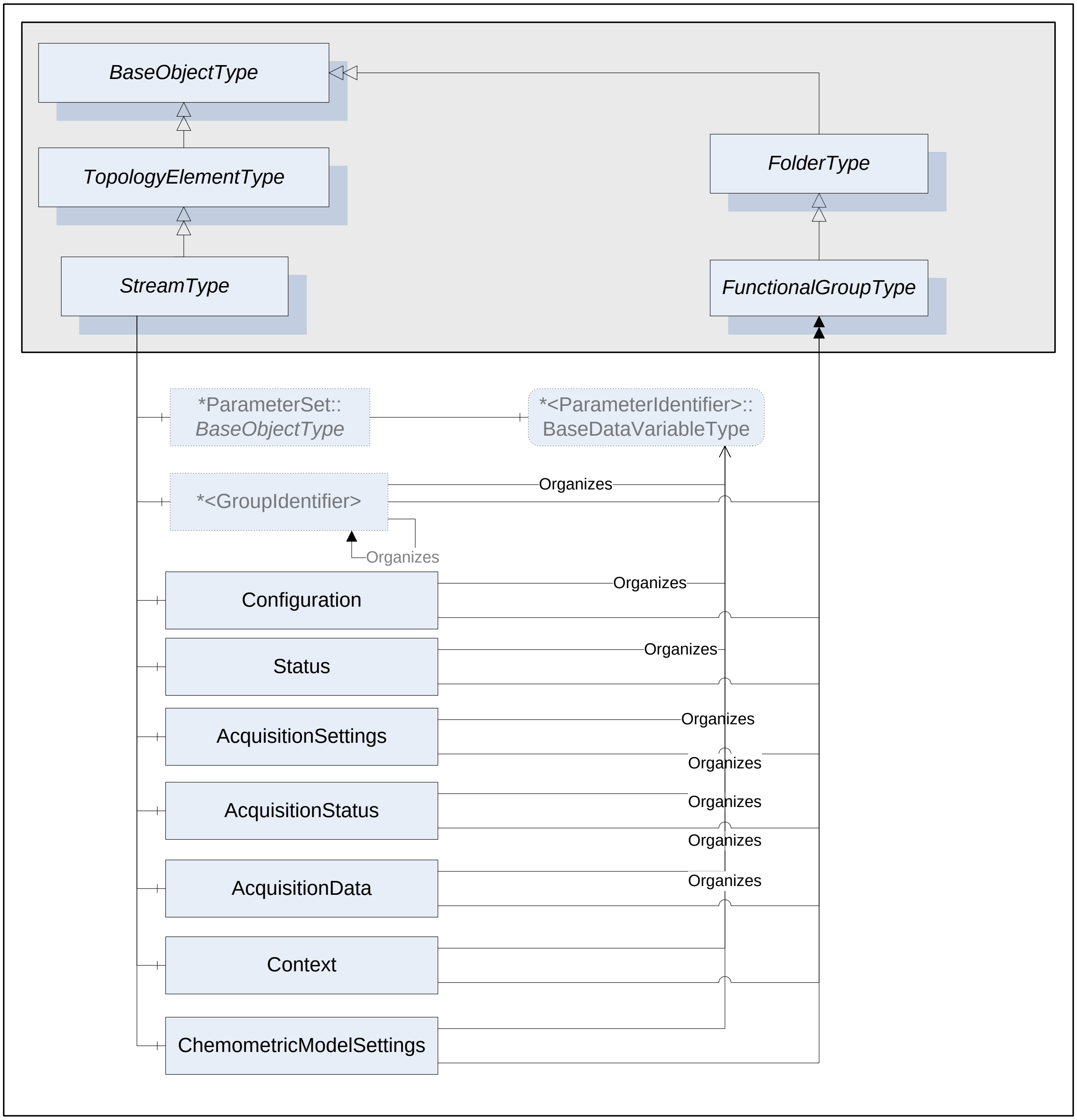

Figure 11 - Stream FunctionalGroups

Table 33 – StreamType Definition

Attribute	Value				
BrowseName	StreamType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *TopologyElementType* defined in [UA-DI].					
HasComponent	Object	ParameterSet		BaseObjectType	Mandatory
HasComponent	Object	<GroupIdentifier>		FunctionalGroupType	OptionalPlaceHolder
HasComponent	Object	Configuration		FunctionalGroupType	Mandatory
HasComponent	Object	Status		FunctionalGroupType	Mandatory
HasComponent	Object	AcquisitionSettings		FunctionalGroupType	Mandatory
HasComponent	Object	AcquisitionStatus		FunctionalGroupType	Mandatory
HasComponent	Object	AcquisitionData		FunctionalGroupType	Mandatory
HasComponent	Object	ChemometricModelSettings		FunctionalGroupType	Mandatory
HasComponent	Object	Context		FunctionalGroupType	Mandatory

5.2.3.2 Parameters of StreamType

Parameters defined for the *StreamType* are described in the following tables. The tables correspond to mandatory *FunctionalGroups* defined for the *StreamType*. Additional *Parameters* may be defined for *Stream* on subtypes of *AnalyserDeviceType* and associated with those *FunctionalGroups*.

All *Stream Parameters* exist as components of the *ParameterSet Object* defined on that *Stream*. Each *Parameter* defined for a *Stream* shall be accessible through one and only one *FunctionalGroup* defined on that *Stream.* Note, that the same *Parameter* is not instantiated more than once. Both, *ParameterSet* and a specific *FunctionalGroup* maintain *Reference*s to the same instance of the *Parameter*.

Table 34 describes the *Parameters* that are organized by the *Configuration FunctionalGroup* of a *Stream*.

Table 34 –Stream *Configuration* Parameters

BrowseName	Description	VariableType	Optional/ Mandatory
IsEnabled	True if this stream maybe used to perform acquisition. This *Parameter* is mainly used for maintenance.	DataItemType (DataType=Boolean)	M
IsForced	True if this *Stream* is forced, which means that is the only *Stream* on this *AnalyserChannel* that can be used to perform acquisitions. This *Parameter* is mainly used for maintenance.	DataItemType (DataType=Boolean)	O

Table 35 describes the *Parameters* that are organized by the *Status FunctionalGroup* of a *Stream.* All *Parameters* organized by this *FunctionalGroup* shall be read-only.

Table 35 –Stream *Status* Parameters

BrowseName	Description	VariableType	Optional/ Mandatory
DiagnosticStatus	Stream health status	*DataItemType* (DataType=DeviceHealthEnumeration)	M
LastCalibrationTime	Time at which the last successful calibration was run. This is the *SourceTimestamp* of the main acquisition data of the first acquisition for this calibration. If unknown, it shall be set to DateTime.MinValue.	DataItemType (DataType=DateTime)	O
LastValidationTime	Time at which the last successful validation was run. This is the *SourceTimestamp* of the main acquisition data of the first acquisition for this validation. If unknown, it shall be set to DateTime.MinValue.	DataItemType (DataType=DateTime)	O
LastSampleTime	Time at which the last sample was acquired. This is the *SourceTimestamp* of the main acquisition data for this sample acquisition. If unknown, it shall be set to DateTime.MinValue.	DataItemType (DataType=DateTime)	M

Table 36 describes the *Parameters* that are organized by the Acquisition*Settings FunctionalGroup* of a *Stream*.

Table 36 - Stream AcquisitionSettings Parameters

BrowseName	Description	VariableType	Optional/ Mandatory
TimeBetweenSamples	Number of milliseconds between two consecutive starts of acquisition. Value 0 means “as fast as possible”	AnalogItemType (DataType=Duration)	O

Table 37 describes the *Parameters* that are organized by the*AcquisitionStatus FunctionalGroup* of a *Stream*. All *Parameters* organized by this *FunctionalGroup* shall be read-only.

Table 37 –Stream *AcquisitionStatus* Parameters

BrowseName	Description	VariableType	Optional/ Mandatory
IsActive	True if this stream is actually running, acquiring data. Only one Stream may be marked as IsActive on a given AnalyserChannel at any given time.	DataItemType (DataType=Boolean)	M
ExecutionCycle	Indicates which acquisition cycle is in progress	DataItemType (ExecutionCycleEnumeration)	M
ExecutionCycleSubcode	Indicates a vendor defined code, which further describes the acquisition cycle.	MultiStateDiscreteType	M
Progress	Indicates the progress of an acquisition (e.g. percentage of completion)	DataItemType (DataType=Float)	M

ExecutionCycle indicates the type of acquisition in progress and it is set in the *SelectExecutionCycle* state of the *AnalyserChannel_OperatingModeExecuteSubStateMachine*..

Progress is a float number from 0 to 100 defining the completion of the ongoing acquisition cycle. The granularity of the *Progress* update is vendor specific. It is set to 0 in the *SelectExecutionCycle* of the *AnalyserChannel_OperatingModeExecuteSubStateMachine*.

Table 38 describes the *Parameters* that are organized by the *AcquisitionData FunctionalGroup* of a *Stream*.

Table 38 –Stream AcquisitionData Parameters

BrowseName	Description	VariableType	Optional/ Mandatory
AcquisitionCounter	Simple counter incremented after each Sampling acquisition performed on this Stream; The counter is not incremented for acquisition cycles other than Sampling. It is used to support detection of missing acquisition. Wrap to 0 when it reaches 2147483647. The starting value at power up is vendor specific	AnalogItemType (DataType=Counter)	M
AcquisitionResultStatus	Quality of the acquisition	DataItemType (AcquisitionResultStatusEnumeration)	M
<ProcessVariableIdentifier>	Most commonly, it is a reference to process data produced as a result of applying the chemometric model to ScaledData.. There can be multiple *Parameters* representing process data and uniquely identified by the <ProcessVariableIdentifier> BrowseName.	ProcessVariableType	O
Offset	The Offset Parameter holds the difference in milliseconds between the start of sample extraction and the start of the analysis.	AnalogItemType (DataType=Duration)	O
Raw Data	Raw data produced as a result of data acquisition on the *Stream* (see definition of raw data)	DataItemType (DataType is defined on a subtype of AnalyserDeviceType)	O
ScaledData*	Scaled data produced as a result of data acquisition on the *Stream* and applying the analyser model. The data type used is analyser dependent. (see definition of scaled data)	DataItemType (DataType is defined on a subtype of AnalyserDeviceType)	M
AcquisitionEndTime	The end time of the AnalyseSample or AnalyseCalibrationSample or AnalyseValidationSample state of the AnalyserChannel_OperatingModeExecuteSubStateMachine state machine. This time should not be used for critical data synchronization but rather for correlation with other external events in the diagnostic context. If unknown, AcquisitionEndTime shall be set to DateTime.MinValue	DataItemType (DataType=DateTime)	M

*Definition of the *ScaledData Parameter* here is only to indicate that this *Parameter* must be defined for a *Stream* on a subtype of an *AnalyserDeviceType*. Since different analyser classes will produce scaled data of different type as their output, it is impossible to fully define this *Parameter* at this level. See *ScaledData Parameter* definition for specific class of analyser. If more than one *ScaledData* is required, *Parameters* representing those additional *ScaledData* shall be called *ScaledData1*, *ScaledData2*... *ScaledData<n>.*

The Offset Parameter holds the difference in milliseconds between the start of sample extraction and the start of the analysis which is *the time in millisconds between the WaitForXXXTrigger to ExtractXXXSample transition and the PrepareXXXSample to AnalyseXXXSample transition.*

As a general rule, a single *Parameter* shall not be used to represent different data elements. For example, *ScaledData* shall be used for the Sample acquisition and another *Parameter* shall be used to publish the output of the Calibration acquisition. However, in the case where the Validation cycle consists only of acquisition of normal samples, the *ScaledData Parameter* may be used. A consumer of data from an Analyser Server must be able to correlate values collected from different *Parameters*. Specifically, it must be possible to associate scaled data with raw data, process data and context data collected during the same acquisition cycle. The data correlation is based on time-stamps used during data collection. *SourceTimestamp* shall be the time when the sampling system starts extracting the sample, defined by the start of the *ExtractSample* or *ExtractCalibrationSample* or *ExtractValidationSample* state of the *AnalyserChannel_OperatingModeExecuteSubStateMachine*. The difference between the *SourceTimestamp* and the time when the sample is analysed, is reflected in the *Offset Parameter defines in AcquisitionData*.

To simplify integration with historians, *Parameters* in the *AcquisitionData FunctionalGroup* shall be updated once per acquisition cycle.

Time-stamp management rules:

1) The time-stamp of the analyser main data (*RawData, ScaledData*) shall be the start time of the ExtractSample or ExtractCalibrationSample or ExtractValidationSample state of the AnalyserChannel_OperatingModeExecuteSubStateMachine.

2) All values derived from acquired data shall have the same *SourceTimestamp* as the acquired data. For example *RawData, ScaledData, AcquisitionEndTime* shall have the same *SourceTimestamp*.

3) If a derived value combines acquired data from different data sources, the time-stamp of the “main” data shall be used. Which data source is the main data, is vendor specific, but shall be consistent and documented.

4) If a derived value combines acquired data from different *AnalyserChannels*, the time-stamp of the “main” *AnalyserChannel* shall be used. Which *AnalyserChannel* is the main *AnalyserChannel*, is vendor specific, but shall be consistent and documented.

5) The last item updated after the end of acquisition (PublishResults state) is *AcquisitionResultStatus* which is set to *GOOD_1, BAD_2, UNKNOWN_3 or PARITAL_4*. This implies that all items that are part of this acquisition shall have been updated; this includes items from *AcquisitionData* and *Context FunctionalGroup*.

6) The OPC UA *SourceTimestamp* is always in UTC time.

For details on *SourceTimestamp* elements of a *DataValue* see [UA part 4].

When the analyser is working in a standalone mode i.e. it is not driven by a DCS or other external control system, the analyser should publish the *Context Parameters* using data provided by user or other system entry system like a barcode reader.

Table 39 describes the *Parameters* that are organized by the *Context FunctionalGroup* of a *Stream*.

Table 39 –Stream Context Parameters

BrowseName	Description	VariableType	Optional/ Mandatory
CampaignId	Defines the current campaign	DataItemType (DataType=String)	O
BatchId	Defines the current batch	DataItemType (DataType=String)	O
SubBatchId	Defines the current sub-batch	DataItemType (DataType=String)	O
LotId	Defines the current lot	DataItemType (DataType=String)	O
MaterialId	Defines the current material	DataItemType (DataType=String)	O
Process	Current Process name	DataItemType (DataType=String)	O
Unit	Current Unit name	DataItemType (DataType=String)	O
Operation	Current Operation name	DataItemType (DataType=String)	O
Phase	Current Phase name	DataItemType (DataType=String)	O
UserId	Login name of the user who is logged on at the device console. If no Operator logon, “System” shall be assigned to UserId.	DataItemType (DataType=String)	O
SampleId	Identifier for the sample	DataItemType (DataType=String)	O

Table 40 shows *Parameters* that will be organized by the *ChemometricModelSettings FunctionalGroup*.

Table 40 – *Stream* ChemometricModelSettings Parameters

BrowseName	Description	VariableType	Optional/ Mandatory
<ChemometricModelId>	Chemometric Model used to convert scaled data into process data	ChemometricModelType (DataType=Byte)	O

5.2.4 Accessory Slot

5.2.4.1 Type definition: AccessorySlotType *ObjectType*

AccessorySlotType defines the general structure of an *AccessorySlot Object*. Figure 12 shows the detailed composition of *AccessorySlotType*. It is formally defined in Table 41.

The *SupportedTypes* folder is used to maintain the set of (sub-types of) *AccessoryTypes* supported by that accessory slot.

AccessorySlotType states are exposed through the *AccessorySlotStateMachine Object* of type *AccessorySlotStateMachineType*. For details on *AccessorySlotStateMachineType* see clause 5.3.5.

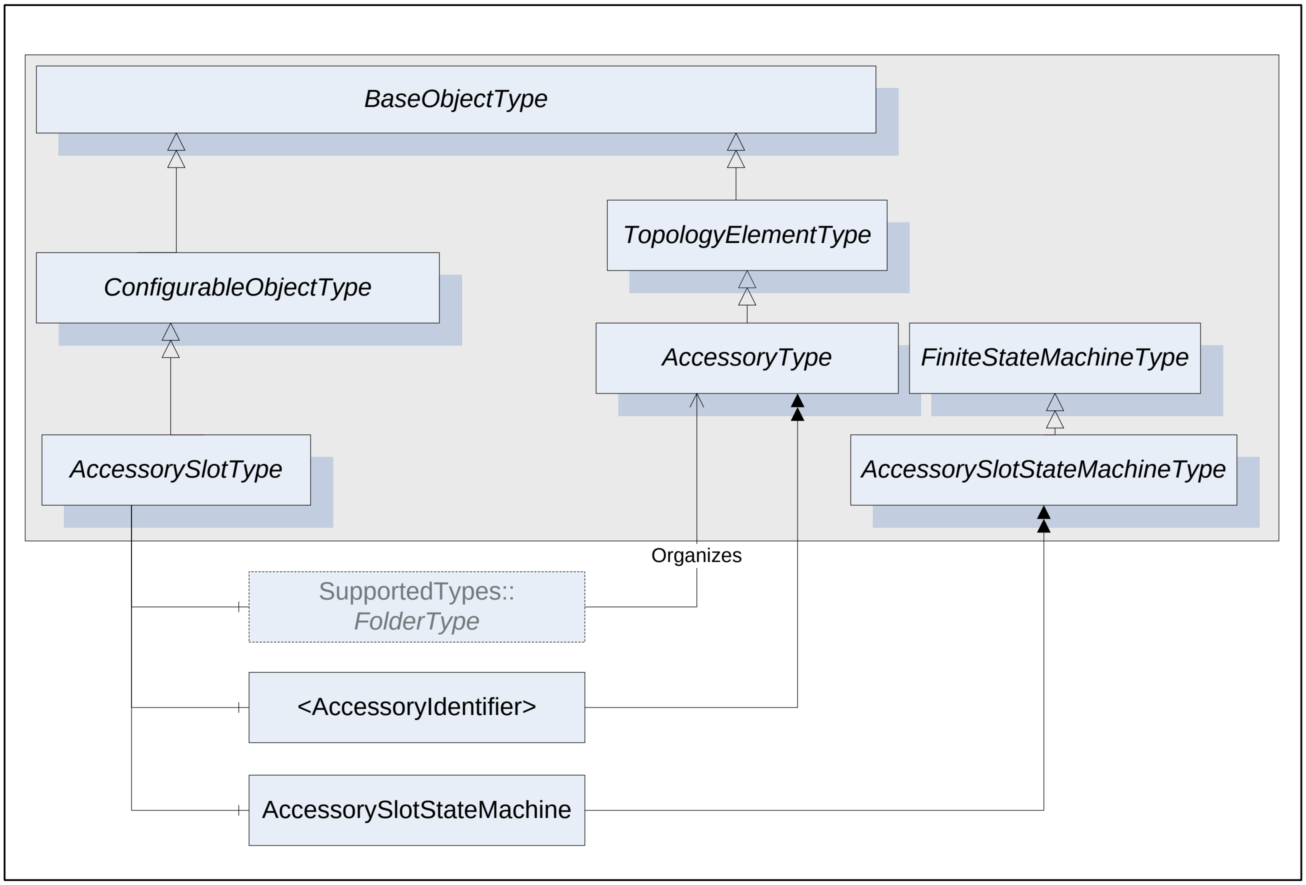

Figure 12 - AccessorySlotType Components

Table 41 – AccessorySlotType Definition

Attribute	**Value**				
BrowseName	AccessorySlotType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *ConfigurableObjectType* defined in [UA DI]					
HasProperty	Variable	IsHotSwappable	Boolean	PropertyType	Mandatory
HasProperty	Variable	IsEnabled	Boolean	PropertyType	Mandatory
HasComponent	Object	AccessorySlotStateMachine		AccessorySlotStateMachineType	Mandatory
HasComponent	Object	<AccessoryIdentifier>		AccessoryType	OptionalPlaceHolder

AccessorySlotType inherits from the ConfigurableObjectType. *SupportedTypes* contain *References* to supported *AccessoryTypes*. .

IsHotSwappable Property is True if an accessory can be inserted in the accessory slot while it is powered.

IsEnabled Property is True if this accessory slot is capable of accepting an accessory in it.

AccessorySlotStateMachine describes internal states of the accessory slot.

<*AccessoryIdentifier*> represents the accessory currently installed in the accessory slot.

5.2.4.2 AccessorySlot *Object*

The term *AccessorySlot* refers to an instance of *AccessorySlotType ObjectType* as defined in Table 41.

AccessorySlotType can be instantiated as components of an *AnalyserDevice Object* or any of its subtypes.

Optionally *AccessorySlotAccessorySlotType* can be instantiated as components of the *AnalyserChannel Objects*.

5.2.5 Accessory

5.2.5.1 Type definition: AccessoryType *ObjectType*

This *ObjectType* defines the structure of an *Accessory Object*. Figure 13 shows the *AccessoryType* components. It is formally defined in Table 42.

AccessoryType is a subtype of *TopologyElementType*.

An Accessory may have *Parameters*. If an Accessory has *Parameters* they appear in an *Object* called *ParameterSet* as a flat list of *Parameters*. *Parameters* of an *Accessory* are identified by <ParameterIdentifier> *Parameters* of an *Accessory* can be organized in *FunctionalGroups* identified as <GroupIdentifier>. An Accessory has at least three *FunctionalGroups* that expose its *Parameters* in an organized fashion. The three mandatory *FunctionalGroups* are:

- *Configuration* - used to organize *Parameters* representing the high-level configuration items of the accessory, which are expected to be modified by end users.
- *Status* - used to organize *Parameters* which describe the general health of the accessory.
- *FactorySettings* - used to organize *Parameters* which describe the factory settings of the accessory and are not expected to be modified by end users.

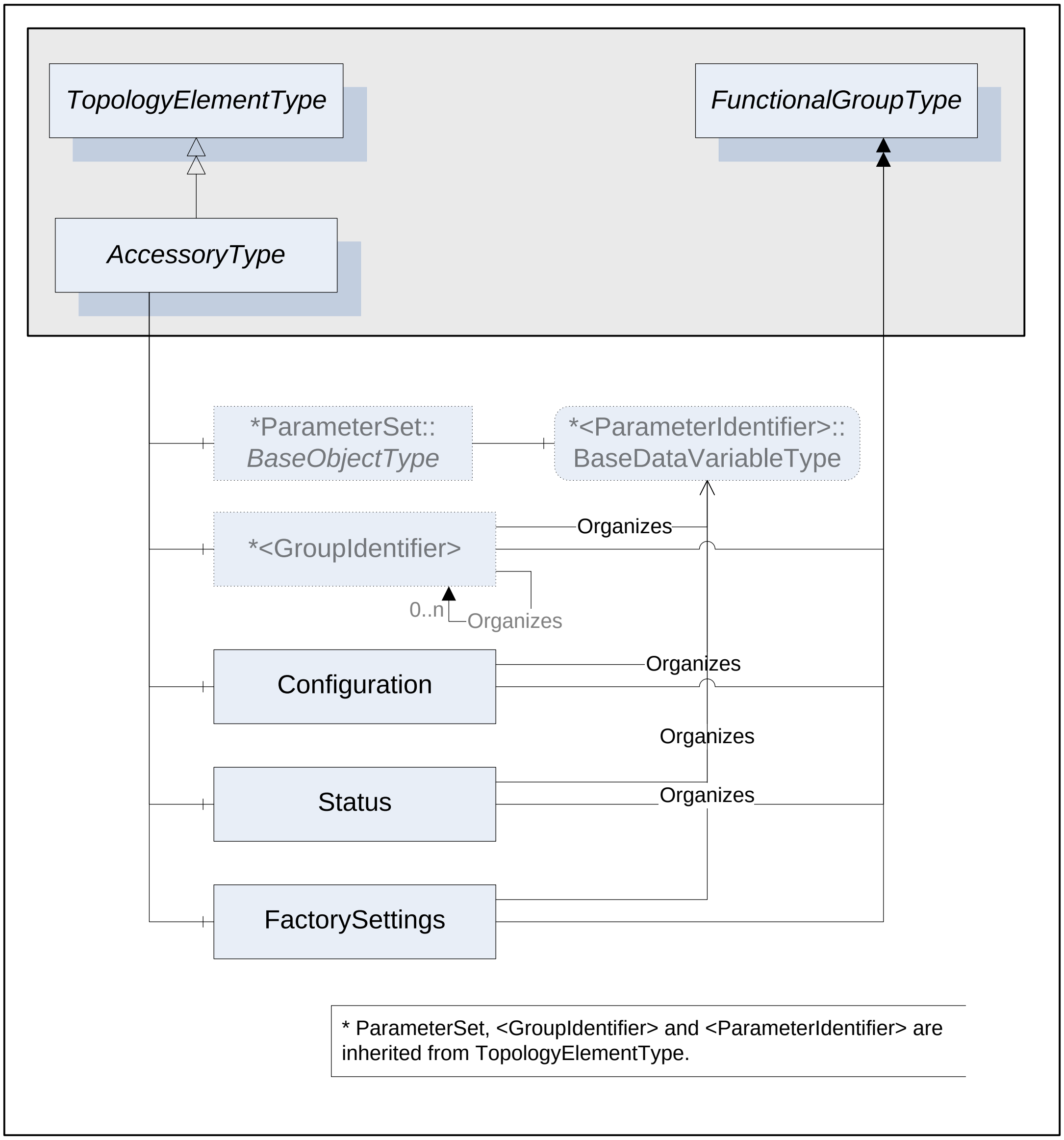

Figure 13 – AccessoryType

Table 42 – AccessoryType Definition

Attribute	**Value**				
BrowseName	AccessoryType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *TopologyElementTypee* defined in [UA-DI]					
HasComponent	Object	Configuration		FunctionalGroupType	Mandatory
HasComponent	Object	Status		FunctionalGroupType	Mandatory
HasComponent	Object	FactorySettings		FunctionalGroupType	Mandatory
HasComponent	Variable	IsHotSwappable	Boolean	PropertyType	Mandatory
HasComponent	Variable	IsReady	Boolean	PropertyType	Mandatory

IsHotSwappable Property is True if this accessory can be inserted in an accessory slot while it is powered. Its value may only be True when it is in *Installed* state. It shall be False in all other states.

IsReady Property is True if this accessory is ready to be used. Its value may only be True when it is in *Installed* state, It shall be False in all other states.

5.2.5.2 Accessory *Object*

The term *Accessory* refers to an instance of *AccessoryType ObjectType* as defined in Table 42.

*Accessory Object*s can be instantiated as components of an AccessorySlot *Object*.

5.2.5.3 Sub-types of AccessoryType *ObjectType*

This specification defines three sub-types of *AccesoryType*: *DetectorType, SmartSamplingSystemType* and *SourceType*.

Table 43 describes a detector *Accessory* which is capable of producing raw data for an analyser.

Table 43 - DetectorType

Attribute	**Value**				
BrowseName	DetectorType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AccessoryType* defined in 5.2.5.					

Table 44 describes an intelligent sampling system *Accessory* used to extract samples from the process monitored by an analyser. It may also be used for non-intrusive device like ATR. It is “smart” in the sense that it provides interaction through configuration and/or status compared to passive sampling systems that provide no status or control capabilities.

Table 44 - SmartSamplingSystemType

Attribute	**Value**				
BrowseName	SmartSamplingSystemType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AccessoryType* defined in 5.2.5.					

Table 45 describes an *Accessory* used by spectrometers (infrared, visible, UV etc.) with internal source that illuminate the sample.

Table 45 - SourceType

Attribute	**Value**				
BrowseName	SourceType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AccessoryType* defined in 5.2.5.					

5.2.6 SpectrometerDevice

5.2.6.1 Type definition: SpectrometerDeviceType *ObjectType*

Table 46 - SpectrometerDeviceType

Attribute	Value				
BrowseName	SpectrometerDeviceType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AnalyserDeviceType* defined in 5.2.1.1					

5.2.6.2 SpectrometerDevice *Object*

The term *SpectrometerDevice* refers to an instance of *SpectrometerDeviceType ObjectType* as defined in Table 46

All *SpectrometerDevice Object*s have *Attributes* and *Properties* that they inherit from the *AnalyserDeviceType*.

5.2.6.3 Parameters of SpectrometerDeviceType

Table 47 describes the *Parameters* that are organized by the *FactorySettings FunctionalGroup* of a *SpectrometerDeviceType*.

Table 47 – Spectrometer*DeviceType* FactorySettings Parameters

BrowseName	Description	VariableType	Optional/ Mandatory
SpectralRange	All spectral ranges that can be covered by this analyser. Vendors are expected to use a subtype of DataItemType to provide engineering units through the standard Property EngineeringUnits of type EUInformation. Typical units will be cm^{-1} and µm.	DataItemType (DataType=Range[])	O

In general, a spectrometer covers one spectral range, but some spectrometers may cover more than one. In case of spectrometers based on a filter wheel, each entry in the array is the band of one of the filters. This is why an array of Range is used as the data type for this *Parameter*.

5.2.6.4 SpectrometerDeviceStreamType

SpectrometerDevice*StreamType* defines seven mandatory *FunctionalGroups* described in5.2.3.1: *Configuration, Status, AcquistionSettings, AcquisitionStatus, AcquisitionData, ChemometricModelSettings, and Context.*

Table 48 describes the *Parameters* that are organized by the *Configuration FunctionalGroup* of a *SpectrometerDeviceStreamType.*

Table 48 – Spectrometer*DeviceStreamType* Configuration Parameters

BrowseName	Description	VariableType	Optional/ Mandatory
ActiveBackground	Background spectrum used for the evaluation of the absorbance. In the case of spectrometer like diode array that requires black and white background, this is the white background.	YArrayItemType (DataType=Float)	M
ActiveBackground1	Background spectrum used for the evaluation of the absorbance. In the case of spectrometer like diode array that requires black and white background, this is the black background and the *Parameter* is mandatory.	YArrayItemType (DataType=Float)	O

If more then one background spectrum is required, *Parameters* representing those additional background spectra shall be called ActiveBackground1, ActiveBackground2,…,ActiveBackground<n> and the same *ModellingRule*s as for ActiveBackground *Parameter* shall apply.

Table 49 describes the *Parameters* that are organized by the *AcquisitionSettings FunctionalGroup* of a *SpectrometerDeviceStreamType.*

Table 49 – Spectrometer*DeviceStreamType* AcquisitionSettings Parameters

BrowseName	Description	VariableType	Optional/ Mandatory
SpectralRange	Spectral range of this acquisition. Vendors are expected to use a subtype of DataItemType to provide engineering units through the standard Property EngineeringUnits of type EUInformation. Typical units will be cm^{-1} and µm.	*DataItemType (DataType=Range)*	O
Resolution	Acquisition resolution May be an enum or Float	DataItemType	O
RequestedNumberOfScans	Number of scans to be averaged This *Parameter* is often referred to as ObservationTime	AnalogItemType (DataType=Int32)	O
Gain	Detector gain May be an enum or Float	DataItemType	O
TransmittanceCutoff	Transmittance clipping limits	DataItemType (DataType=Range)	O
AbsorbanceCutoff	Absorbance clipping limits	DataItemType (DataType=Range)	O

Many of the *Parameters* in the *AcquisitionSettings FunctionalGroup* are used for sample acquisition. Calibration and validation may or may not use the same value. It is up to the vendor to select his approach: share *Parameters* or use different ones. Nested *FunctionalGroup* may also be used to organize different set of *Parameters.*

Table 50 describes the *Parameters* that are organized by the *AcquisitionStatus FunctionalGroup* of a *SpectrometerDeviceStreamType.* All *Parameters* organized by this *FunctionalGroup* shall be read-only.

Table 50 – Spectrometer*DeviceStreamType* AcquisitionStatus Parameters

BrowseName	Description	VariableType	RW	Optional/ Mandatory
NumberOfScansDone	Actual number of scans completed	AnalogItemType (DataType=Int32)	RO	O

Table 51 describes the *Parameters* that are organized by the *AcquisitionData FunctionalGroup* of a *SpectrometerDeviceStreamType.*

Table 51 – Spectrometer*DeviceStreamType* AcquisitionData Parameters

BrowseName	Description	VariableType	RW	Optional/ Mandatory
Raw Data	Raw spectrum in arbitrary units	YArrayItemType (DataType=Float)	RO	O
ScaledData*	Absorbance	YArrayItemType (DataType=Float)	RO	M
TotalNumberOfScansDone	Total number of scans done at the end of acquisition.	AnalogItemType (DataType=Int32)	RO	M
BackgroundAcquisitionTime	Time stamp of the background used for this acquisition. If more then one background spectrum is required, the time of ActiveBackground shall be used. Background is acquired during calibration acquisition cycle.	DataItemType (DataType=DateTime)	RO	M
PendingBackground	Last acquired Background spectrum. This Background is not automatically used for evaluation of ScaledData (Absorbance) - see ActiveBackground *Parameter*. In the case of spectrometer like diode array that requires black and white background, this is the white background.	YArrayItemType (DataType=Float)	RO	M
PendingBackground1	Last acquired Background spectrum. This Background is not automatically used for evaluation of ScaledData (Absorbance) - see ActiveBackground *Parameter*. In the case of spectrometer like diode array that requires black and white background, this is the black background and the *Parameter* is mandatory	YArrayItemType (DataType=Float)	RO	O

If more then one background spectrum is required, *Parameters* representing those additional background spectra shall be called PendingBackground1, PendingBackground2,...,PendingBackground<n> and the same *ModellingRules* as for PendingBackground *Parameter* shall apply.

* ScaledData *Parameter* at this level represents the same *Parameter* that was defined on *StreamType*. Since different types of analysers may represent ScaledData differently, it was impossible to declare the *VariableType* of this *Parameter* at the *StreamType* level. It is possible here because the scope of the definition is limited to *SpectrometerDeviceType*. Devices of this type use *YArrayItemType* to represent ScaledData.

5.2.7 MassSpectrometerDevice

5.2.7.1 Type definition: MassSpectrometerDeviceType *ObjectType*

Table 52 - MassSpectrometerDeviceType

Attribute	Value				
BrowseName	MassSpectrometerDeviceType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AnalyserDeviceType* defined in 5.2.1.1					

5.2.7.2 MassSpectrometerDevice *Object*

The term *MassSpectrometerDevice* refers to an instance of *MassSpectrometerDeviceType ObjectType* as defined in Table 52.

5.2.7.3 MassSpectrometerDeviceStreamType

There is no specific Parameter in MassSpectrometerDeviceStreamType.

5.2.8 ParticleSizeMonitorDevice

5.2.8.1 Type definition: ParticleSizeMonitorDeviceType *ObjectType*

Particle size can be determined by light scattering (e.g. Focus Beam Reflectance Measurement, Laser Diffraction) or other *Method*s. This type of analyser can be used to implement particle monitoring technique for in-line real-time measurement of particle size. A wide range of industrial process control applications are therefore possible such as the online control of crystallizers.

ParticleSizeMonitorDeviceType defines the general structure of a *ParticleSizeMonitorDevice Object.*

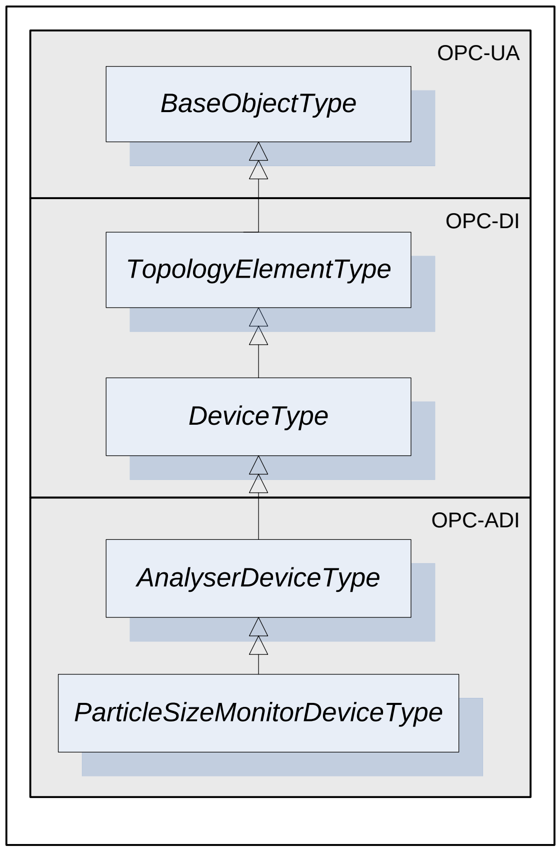

Figure 14 - ParticleSizeMonitorDeviceType

ParticleSizeMonitorDeviceType is a subtype of *AnalyserDeviceType.*

Table 53 - ParticleSizeMonitorDeviceType

Attribute	Value				
BrowseName	ParticleSizeMonitorDeviceType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AnalyserDeviceType* defined in 5.2.1.1					

5.2.8.2 ParticleSizeMonitorDevice *Object*

The term *ParticleSizeMonitorDevice* refers to an instance of *ParticleSizeMonitorDeviceType ObjectType* as defined in Table 53.

All *ParticleSizeMonitorDevice* have *Attributes* and *Properties* that they inherit from the *AnalyserDeviceType*.

5.2.8.3 ParticleSizeMonitorDeviceStreamType

ParticleSizeMonitorDevice*StreamType* defines seven mandatory *FunctionalGroups* described in5.2.3.1: *Configuration*, *Status*, *AcquistionSettings*, *AcquisitionStatus*, *AcquisitionData*, *ChemometricModelSettings*, *Context. Parameters* exposed by an *Stream* of a *ParticleSizeMonitorDevice* should be organized by those *FunctionalGroups* based on their meaning.

Table 54 describes the *Parameters* that are organized by the *AcquisitionData FunctionalGroup* of a *ParticleSizeMonitorDeviceStreamType.*

Table 54 – *ParticleSizeMonitorDeviceStreamType* AcquisitionData Parameters

BrowseName	Description	VariableType	Optional/ Mandatory
Background	Array describing the measured background on detector(s.)	YArrayItemType (DataType=Float)	O
Raw Data	Array describing the measured raw data on detector(s) in arbitrary units.	YArrayItemType (DataType=Float)	O
ScaledData	Array describing the corrected measured data detector(s), for example after background subtraction	YArrayItemType (DataType=Float)	M
SizeDistribution	Returns the Particle Size Distribution	YArrayItemType (DataType=Float)	M
BackgroundAcquisitionTime	Time stamp of the background used for this acquisition	DataItemType (DataType=DateTime)	M

5.2.9 AcousticSpectrometerDevice

5.2.9.1 Type definition: AcousticSpectrometerDeviceType *ObjectType*

Table 55 - AcousticSpectrometerDeviceType

Attribute	Value				
BrowseName	AcousticSpectrometerDeviceType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AnalyserDeviceType* defined in 5.2.1.1					

5.2.9.2 AcousticSpectrometerDevice *Object*

The term *AcousticSpectrometerDevice* refers to an instance of *AcousticSpectrometerDeviceType ObjectType* as defined in Table 55.

5.2.9.3 AcousticSpectrometerDeviceStreamType

There is no specific Parameter in AcousticSpectrometerDeviceStreamType.

5.2.10 ChromatographDevice

5.2.10.1 Type definition: ChromatographDeviceType *ObjectType*

Chromatograph retrieves the concentration of chemical components by using a set of separation columns that separate each molecule based on the time it takes to go through a given column path.

ChromatographrDeviceType defines the general structure of a *ChromatographDevice Object*

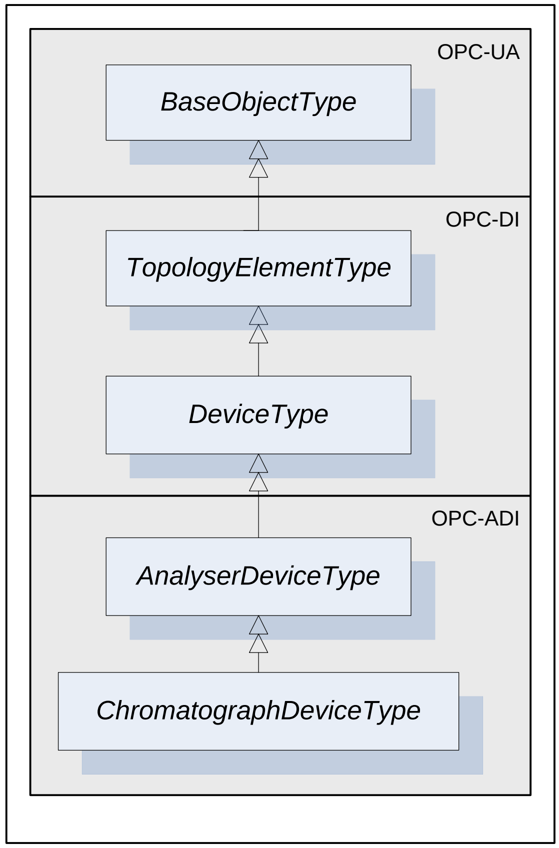

Figure 15 - ChromatographDeviceType

ChromatographDeviceType is a subtype of *AnalyserDeviceType*

Table 56 - ChromatographDeviceType

Attribute	Value				
BrowseName	ChromatographDeviceType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AnalyserDeviceType* defined in 5.2.1.1					

5.2.10.2 ChromatographDevice *Object*

The term *ChromatographDevice* refers to an instance of *ChromatographType ObjectType* as defined in Table 56.

All *ChromatographDevices* have *Attributes* and *Properties* that they inherit from the *AnalyserDeviceType*.

5.2.10.3 ChromatographDeviceStreamType

StreamType defines seven mandatory *FunctionalGroups* described in5.2.3.1: *Configuration, Status, AcquistionSettings, AcquisitionStatus, AcquisitionData, ChemometricModelSettings, and Context.* The following tables describe *Parameters* defined on the *Stream* of a *ChromatographDevice*.

Table 40 describes the *Parameters* that are organized by the *AcquisitionData FunctionalGroup* of a *ChromatographDeviceStreamType.*

Table 57 – Chromatograph*DeviceStreamType* AcquisitionData Parameters

BrowseName	Description	VariableType	Optional/ Mandatory
ScaledData*	Chromatogram	YArrayItemType [] (DataType=Float)	M
ComponentX	Component analysed by a chromatograph	EngineeringValueType (DataType=Float)	M

* ScaledData *Parameter* at this level represents the same *Parameter* that was defined on *StreamType*. Since different types of analysers may represent ScaledData differently, it was impossible to declare the *VariableType* of this *Parameter* at the *StreamType* level. It is possible here because the scope of the definition is limited to *ChromatographDeviceType*. Devices of this type use array of *YArrayItemType* to represent ScaledData.

The YArrayItem describing the chromatogram has the following behaviors:

- Because the Chromatograph may collect many chromatograms simultaneously, *ScaledData* is an array of *YArrayItem*.
- X axis is the time in seconds since the injection time, which is the start of the ExtractSample or ExtractCalibrationSample or ExtractValidationSample state of the AnalyserChannel_OperatingModeExecuteSubStateMachine*.*
- Y axis unit is vendor specific, usually volts at the detector output.
- To reduce data bandwidth, the X axis may not be continuous i.e. when there is no peak, no data is produced. This implies that the *xAxisDefinition.axisSteps* shall be provided.
- The *xAxisDefinition.axisSteps* of each chromatogram may be different because the peak positions are different from column to column.

5.2.10.4 Component

The Chromatograph Component values are mapped using EngineeringValueType and they are placed under the appropriate *Stream* in the *AcquisitionData FunctionalGroup*. Annex B provides an example of its sub-elements.

5.2.10.5 GCOvenType

Table 58 describes a gas chromatograph oven *Accessory* which maintains its set of valves, columns and detectors at the temperature defined by the chromatographic application.

Table 58 - GCOvenType

Attribute	Value				
BrowseName	GCOvenType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AccessoryType* defined in 5.2.5.					

5.2.11 NMRDevice

5.2.11.1 Type definition: NMRDeviceType *ObjectType*

Table 59 - NMRDeviceType

Attribute	Value				
BrowseName	NMRDeviceType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AnalyserDeviceType* defined in 5.2.1.1					

5.2.11.2 NMRDevice *Object*

The term *NMRDevice* refers to an instance of *NMRDeviceType ObjectType* as defined in Table 59.

5.2.11.3 NMRDeviceStreamType

There is no specific Parameter in NMRStreamType.

5.3 State Machines

5.3.1 Introduction

The following diagram shows the state and command model for the subclasses of the *AnalyserDeviceType, AnalyserChannelType* and *AccessorySlotType*. An *AnalyserDeviceType* contains a state machine of type *AnalyserDeviceStateMachineType*. *AnalyserChannelType* contains a state machine of type *AnalyserChannelStateMachineType*. *AccessorySlotType* contains a state machine of type *AccessorySlotStateMachineType*. (See [UA Part 5] Appendix B for a description of state machines.)

For all state machines defined in this specification, for each self-*Transition* (where the from-state and to-state are the same) that is used to indicate the progress within a state, the self-*Transition* shall occur only if the time required to pass through this state exceeds 5 seconds and shall reoccur at 5 (±1) second intervals. The T*ransition* event should include information on the remaining time to complete this state when available

All state machines defined in this specification are mandatory unless explicitly stated otherwise. However, some states may be implemented as transient (do-nothing) states depending on the unique characteristics of an analyser.

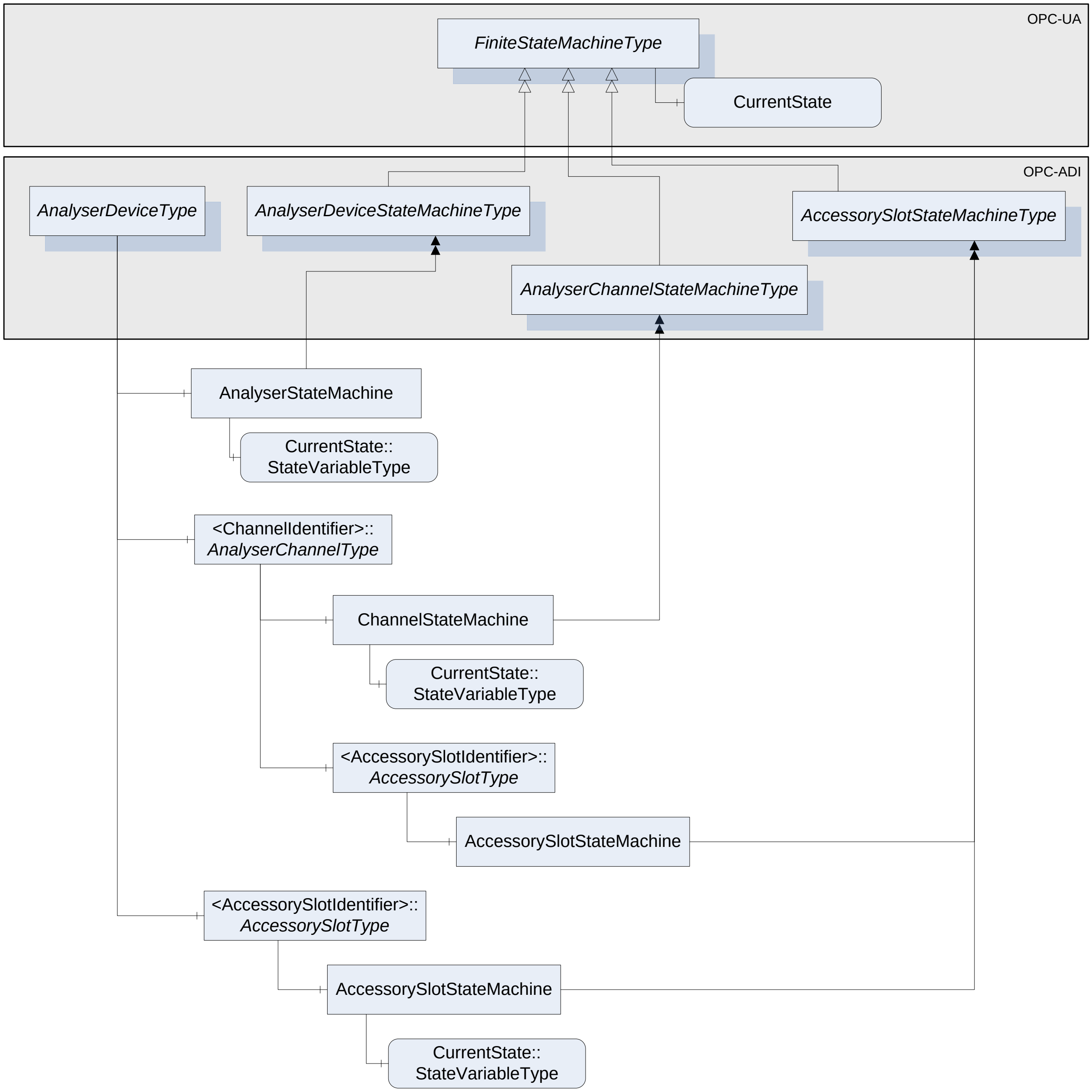

Figure 16 - ADI State Machines

5.3.2 AnalyserDeviceStateMachineType

AnalyserDeviceStateMachineType is a subtype of *FiniteStateMachineType*. The states are derived from the *ANSI/ISA TR 88.02-2008 Machine and Unit States* Technical Report [ISA-88 TR], which in turn were derived from the OMAC PackML tag definition set and the ANSI/ISA 88 Part 1 standard [ISA-88].

AnalyserDeviceStateMachineType contains a nested state model that defines the top level states Operating, Local and Maintenance (called Modes in [ISA-88 TR] and OMAC) of a device.

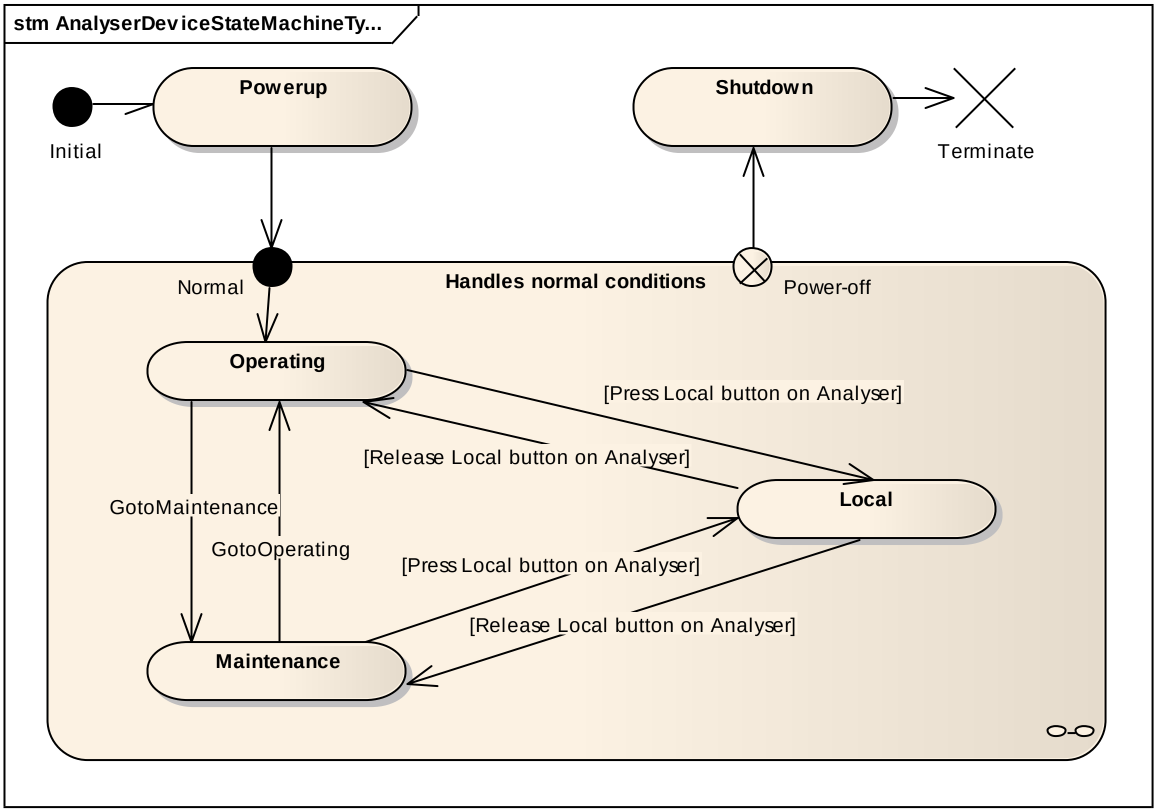

Figure 17 - AnalyserDeviceStateMachine

The *Powerup* state is where the *AnalyserDevice* waits for the completion of the power-up setup. Its sub-states are out of scope of the ADI specification.

The *Shutdown* state is where the *AnalyserDevice* waits for the completion of the power down sequence. Its sub-states are out of scope of the ADI specification.

5.3.2.1 Type definition: AnalyserDeviceStateMachineType *ObjectType*

AnalyserDeviceStateMachineType.is formally defined in Table 60 .

Table 60 – AnalyserDeviceStateMachineType Definition

Attribute	Value				
BrowseName	AnalyserDeviceStateMachineType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *FiniteStateMachineType* defined in [UA Part 5]					
HasComponent	Object	Powerup		InitialStateType	Mandatory
HasComponent	Object	Operating		StateType	Mandatory
HasComponent	Object	Local		StateType	Mandatory
HasComponent	Object	Maintenance		StateType	Mandatory
HasComponent	Object	Shutdown		StateType	Mandatory
HasComponent	Object	PowerupToOperatingTransition		TransitionType	Mandatory
HasComponent	Object	OperatingToLocalTransition		TransitionType	Mandatory
HasComponent	Object	OperatingToMaintenanceTransition		TransitionType	Mandatory
HasComponent	Object	LocalToOperatingTransition		TransitionType	Mandatory
HasComponent	Object	LocalToMaintenanceTransition		TransitionType	Mandatory
HasComponent	Object	MaintenanceToOperatingTransition		TransitionType	Mandatory
HasComponent	Object	MaintenanceToLocalTransition		TransitionType	Mandatory
HasComponent	Object	OperatingToShutdownTransition		TransitionType	Mandatory
HasComponent	Object	LocalToShutdownTransition		TransitionType	Mandatory
HasComponent	Object	MaintenanceToShutdownTransition		TransitionType	Mandatory

5.3.2.2 AnalyserDeviceStateMachineType States

5.3.2.2.1 Introduction

Table 61 specifies the *AnalyserStateMachine's* State *Objects*. These State *Objects* are instances of the *StateType* defined in [UA Part 5] – Appendix B. Each State is assigned a unique *StateNumber* value. Subtypes of the *AnalyserDeviceStateMachineType* can add *References* from any state to a subordinate or nested *StateMachine Object* to extend the *FiniteStateMachine*.

See Table 62 for a description of the states.

Table 61 – AnalyserDeviceStateMachineType States

BrowseName	References	Target BrowseName	Value	Target Type Definition	Notes
States					
Powerup	HasProperty	StateNumber	100	PropertyType	
	ToTransition	PowerupToOperatingTransition		TransitionType	
Operating	HasProperty	StateNumber	200	PropertyType	
	FromTransition	PowerupToOperatingTransition		TransitionType	
	FromTransition	MaintenanceToOperatingTransition		TransitionType	
	FromTransition	LocalToOperatingTransition		TransitionType	
	ToTransition	OperatingToLocalTransition		TransitionType	
	ToTransition	OperatingToMaintenanceTransition		TransitionType	
	ToTransition	OperatingToShutdownTransition		TransitionType	
Local	HasProperty	StateNumber	300	PropertyType	
	FromTransition	OperatingToLocalTransition		TransitionType	
	FromTransition	MaintenanceToLocalTransition		TransitionType	
	ToTransition	LocalToOperatingTransition		TransitionType	
	ToTransition	LocalToMaintenanceTransition		TransitionType	
	ToTransition	LocalToShutdownTransition		TransitionType	
Maintenance	HasProperty	StateNumber	400	PropertyType	
	FromTransition	OperatingToMaintenanceTransition		TransitionType	
	FromTransition	LocalToMaintenanceTransition		TransitionType	
	ToTransition	MaintenanceToOperatingTransition		TransitionType	
	ToTransition	MaintenanceToLocalTransition		TransitionType	
	ToTransition	MaintenanceToShutdownTransition		TransitionType	
Shutdown	HasProperty	StateNumber	500	PropertyType	
	FromTransition	OperatingToShutdownTransition		TransitionType	
	FromTransition	LocalToShutdownTransition		TransitionType	
	FromTransition	MaintenanceToShutdownTransition		TransitionType	

A standard set of states are defined for analyser devices. These states represent the operational condition of the device. All devices that contain an *AnalyserDeviceStateMachineType* must support this base set. A device may or may not require a *Client* action to cause the state to change, as defined in the state descriptions below.

Table 62 – AnalyserDeviceStateMachineType State Description

StateName	Description
Powerup	The AnalyserDevice is in its power-up sequence and cannot perform any other task.
Operating	The AnalyserDevice is in the Operating mode. The ADI *Client* uses this mode for normal operation: configuration, control and data collection. In this mode, each child AnalyserChannels are free to accept commands from the ADI *Client* and the *Parameter* values published in the address space values are expected to be valid. When entering this state, all AnalyserChannels of this AnalyserDevice automatically leave the SlaveMode state and enter their Operating state.
Local	The AnalyserDevice is in the Local mode. This mode is normally used to perform local physical maintenance on the analyser. To enter the Local mode, the operator shall push a button, on the analyser itself. This may be a physical button or a graphical control on the local console screen. To quit the Local mode, the operator shall press the same or another button on the analyser itself. When the analyser is in Local mode, all child AnalyserChannels sit in the SlaveMode state of the AnalyserChannelStateMachine. In this mode, no commands are accepted from the ADI interface and no guarantee is given on the values in the address space.

StateName	Description
Maintenance	The AnalyserDevice is in the Maintenance mode. This mode is used to perform remote maintenance on the analyser like firmware upgrade. To enter in Maintenance mode, the operator shall call the GotoMaintenance *Method* from the ADI *Client*. To return to the Operating mode, the operator shall call the GotoOperating *Method* from the ADI *Client*. When the analyser is in the Maintenance mode, all child AnalyserChannels sit in the SlaveMode state of the AnalyserChannelStateMachine. In this mode, no commands are accepted from the ADI interface for the AnalyserChannels and no guarantee is given on the values in the address space.
Shutdown	The AnalyserDevice is in its power-down sequence and cannot perform any other task.

The set of states defined to describe an AnalyserDevice can be expanded. Sub-states can be defined for the base states to provide more resolution to the process and to describe the cause and effects of additional stimuli and transitions.

5.3.2.2.2 Operating State

The Operating state of the *AnalyserDeviceStateMachineType* has no required sub-states.

5.3.2.2.3 Local State

The Local state of the *AnalyserDeviceStateMachineType* has no required sub-states.

The Local state provides suitably authorized personnel the ability to operate individual subordinate equipment controls (such as accessory logic) within the device under manual control (often pushbutton or embedded HMI). Such controls in this state may be on a "hold-to-run" basis such that removal of the run signal will cause a device to be stopped. The ability to perform specific functions will be dependent upon mechanical constraints and interlocks. Local state may be of particular use for setting up the machine to work.

5.3.2.2.4 Maintenance State

The Maintenance state of the *AnalyserDeviceStateMachineType* has no required sub-states.

The Maintenance state allows suitably authorized personnel the ability to run an individual device independent of other devices that may be in the same production line or lab cell. This would typically be used for faultfinding, device trials or testing operational improvements.

5.3.2.3 AnalyserDeviceStateMachineType Transitions

Transitions are instances of *Objects* of the *TransitionType* defined in [UA Part 5] – Appendix B which also includes the definitions of the ToState, FromState, HasCause, and HasEffect *References* used. Table 63 specifies the Transitions defined for the *AnalyserDeviceStateMachineType.* Each Transition is assigned a unique *TransitionNumber*.

Table 63 – AnalyserDeviceStateMachineType Transitions

BrowseName	References	Target BrowseName	Value	Target Type Definition	Notes
Transitions					
PowerupToOperatingTransition	HasProperty	TransitionNumber	1	PropertyType	
	FromState	Powerup		InitialStateType	
	ToState	Operating		StateType	
	HasCause	Analyser is powering-up			External cause
OperatingToLocalTransition	HasProperty	TransitionNumber	2	PropertyType	
	FromState	Operating		StateType	
	ToState	Local		StateType	
	HasCause	Pressing Local button on analyser			External cause
OperatingToMaintenanceTransition	HasProperty	TransitionNumber	3	PropertyType	
	FromState	Operating		StateType	
	ToState	Maintenance		StateType	
	HasCause	GotoMaintenance		Method	
LocalToOperatingTransition	HasProperty	TransitionNumber	4	PropertyType	
	FromState	Local		StateType	
	ToState	Operating		StateType	
	HasCause	Releasing Local button on analyser			External cause
LocalToMaintenanceTransition	HasProperty	TransitionNumber	5	PropertyType	
	FromState	Local		StateType	
	ToState	Maintenance		StateType	
	HasCause	Releasing Local button on analyser			External cause
MaintenanceToOperatingTransition	HasProperty	TransitionNumber	6	PropertyType	
	FromState	Maintenance		StateType	
	ToState	Operating		StateType	
	HasCause	GotoOperating		Method	
MaintenanceToLocalTransition	HasProperty	TransitionNumber	7	PropertyType	
	FromState	Maintenance		StateType	
	ToState	Local		StateType	
	HasCause	Pressing Local button on analyser			External cause
OperatingToShutdownTransition	HasProperty	TransitionNumber	8	PropertyType	
	FromState	Operating		StateType	
	ToState	Shutdown		StateType	
	HasCause	Analyser is powering-down			External cause
LocalToShutdownTransition	HasProperty	TransitionNumber	9	PropertyType	
	FromState	Local		StateType	
	ToState	Shutdown		StateType	
	HasCause	Analyser is powering-down			External cause
MaintenanceToShutdownTransition	HasProperty	TransitionNumber	10	PropertyType	
	FromState	Maintenance		StateType	
	ToState	Shutdown		StateType	
	HasCause	Analyser is powering-down			External cause

5.3.3 AnalyserChannelStateMachineType

5.3.3.1 Introduction

AnalyserChannelStateMachineType is a subtype of *FiniteStateMachineType*. The states are derived from the *ANSI/ISA TR 88.02-2008 Machine and Unit States* Technical Report [ISA-88 TR], which in turn were derived from the OMAC PackML tag definition set and the ANSI/ISA 88 Part 1 standard [ISA-88].

AnalyserChannelStateMachineType contains a nested state model that defines the top level states Operating, Local and Maintenance (called Modes in [ISA-88 TR] and OMAC) and the Operating sub-states of a device.

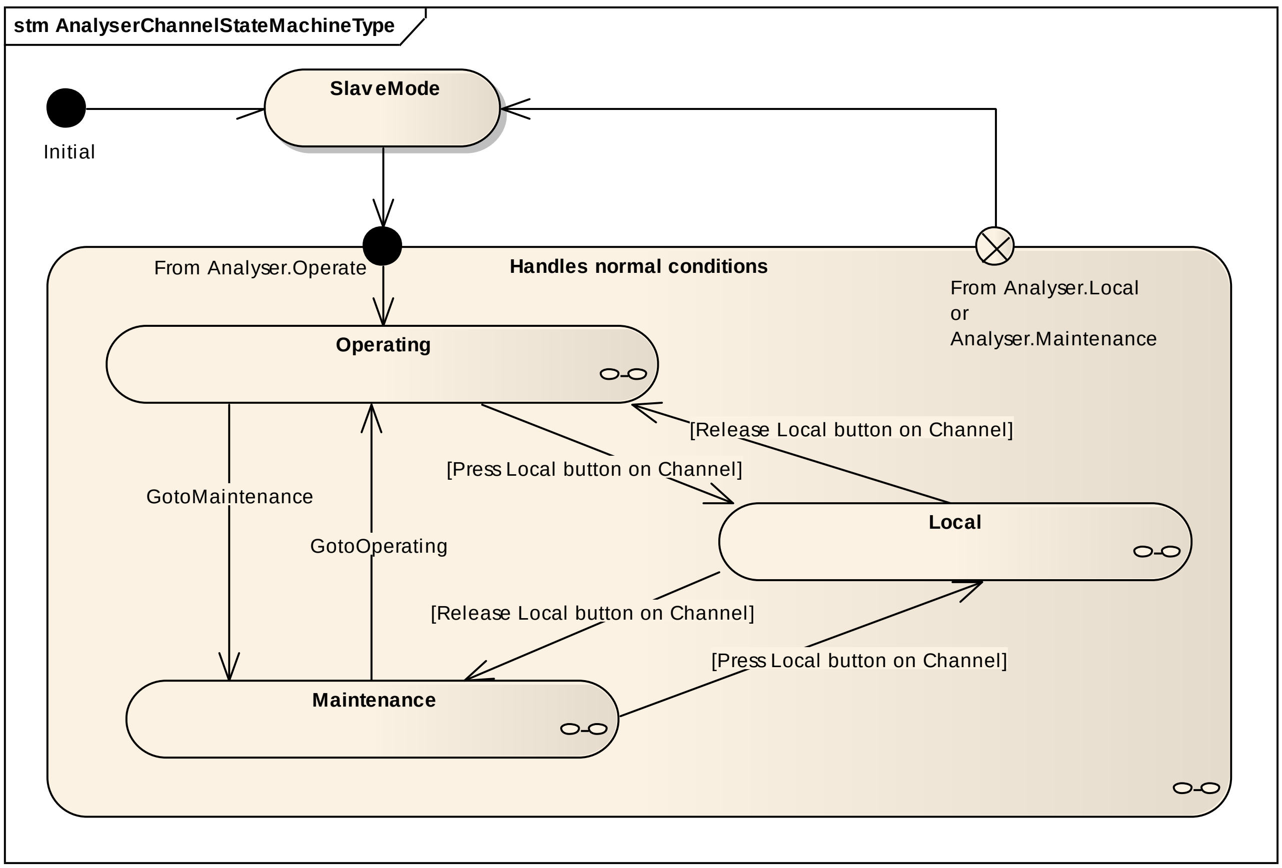

Figure 18 - AnalyserChannelStateMachine

The *SlaveMode* state is where the *AnalyserChannel* stays when its parent *AnalyserDevice* is in Local or Maintenance mode. In this context, the *AnalyserDevice* has the absolute control over all of its *AnalyserChannels*.

The Local button refers to a Local button on a given analyser channel for symmetry with the analyser device.

5.3.3.2 Type definition: AnalyserChannelStateMachineType *ObjectType*

AnalyserChannelStateMachineType.is formally defined in Table 64.

Table 64 – AnalyserChannelStateMachineType Definition

Attribute	Value				
BrowseName	AnalyserChannelStateMachineType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**Data Type**	**TypeDefinition**	**Modelling Rule**
Subtype of the *FiniteStateMachineType* defined in [UA Part 5]					
HasComponent	Object	SlaveMode		InitialStateType	Mandatory
HasComponent	Object	Operating		AnalyserChannelOperatingStateType	Mandatory
HasComponent	Object	Local		AnalyserChannelLocalStateType	Mandatory
HasComponent	Object	Maintenance		AnalyserChannelMaintenanceStateType	Mandatory
HasComponent	Object	SlaveModeToOperatingTransition		TransitionType	Mandatory
HasComponent	Object	OperatingToLocalTransition		TransitionType	Mandatory
HasComponent	Object	OperatingToMaintenanceTransition		TransitionType	Mandatory
HasComponent	Object	LocalToOperatingTransition		TransitionType	Mandatory
HasComponent	Object	LocalToMaintenanceTransition		TransitionType	Mandatory
HasComponent	Object	MaintenanceToOperatingTransition		TransitionType	Mandatory
HasComponent	Object	MaintenanceToLocalTransition		TransitionType	Mandatory
HasComponent	Object	OperatingToSlaveModeTransition		TransitionType	Mandatory
HasComponent	Object	LocalToSlaveModeTransition		TransitionType	Mandatory
HasComponent	Object	MaintenanceToSlaveModeTransition		TransitionType	Mandatory
HasComponent	Object	OperatingSubStateMachine		AnalyserChannel_OperatingModeSubStateMachineType	Mandatory
HasComponent	Object	LocalSubStateMachine		FiniteStateMachineType	Optional
HasComponent	Object	MaintenanceSubStateMachine		FiniteStateMachineType	Optional

GotoOperating Method transitions the *AnalyserChannel* to Operating mode.

GotoMaintenance Method transitions the *AnalyserChannel* to Maintenance mode.

Table 65 – AnalyserChannelOperatingStateType Definition

Attribute	Value				
BrowseName	AnalyserChannelOperatingStateType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**Data Type**	**TypeDefinition**	**Modelling Rule**
Subtype of the *StateType* defined in [UA Part 5]					
HasSubStateMachine	Object	OperatingSubStateMachine		AnalyserChannel_OperatingModeSubStateMachineType	Mandatory

Table 66 – AnalyserChannelLocalStateType Definition

Attribute	Value				
BrowseName	AnalyserChannelLocalStateType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModelingRule**
Subtype of the *StateType* defined in [UA Part 5]					
HasSubStateMachine	Object	LocalSubStateMachine		FiniteStateMachineType	Optional

Table 67 – AnalyserChannelMaintenanceStateType Definition

Attribute	Value				
BrowseName	AnalyserChannelMaintenanceStateType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**Data Type**	**TypeDefinition**	**Modelling Rule**
Subtype of the *StateType* defined in [UA Part 5]					
HasSubStateMachine	Object	MaintenanceSubStateMachine		FiniteStateMachineType	Optional

5.3.3.3 AnalyserChannelStateMachineType States

Table 69 specifies the *AnalyserChannelStateMachine's* State *Objects*. These State *Objects* are instances of the *StateType* defined in [UA Part 5] – Appendix B. Each State is assigned a unique *StateNumber* value. Subtypes of the *AnalyserChannelStateMachineType* can add *References* from any state to a subordinate or nested *StateMachine Object* to extend the *FiniteStateMachine*.

A standard set of states are defined for analyser channels. These states represent the operational condition of the channel. All devices that contain an *AnalyserChannelStateMachineType* shall support this base set. A channel may or may not require a client action to cause the state to change. See Table 68 for a description of the states.

Table 68 – AnalyserChannelStateMachineType State Description

StateName	Description
SlaveMode	The AnalyserDevice is in Local or Maintenance mode and all AnalyserChannels are in SlaveMode
Operating	The AnalyserChannel is in the Operating mode. The ADI *Client* uses this mode for normal operation: configuration, control and data collection. In this mode, AnalyserChannel can accept commands from the ADI *Client* and the *Parameters* published in the address space values are expected to be valid.

StateName	Description
Local	The AnalyserChannel is in the Local mode. This mode is normally used to perform local physical maintenance on the AnalyserChannel. To enter the Local mode, the operator shall push a button, on the AnalyserChannel itself. This may be a physical button or a graphical control on the local console screen. To quit the Local mode, the operator shall press the same or another button on the AnalyserChannel itself. When the AnalyserChannel is in the Local mode, the parent AnalyserDevice has no control over it. In this mode, no commands are accepted from the ADI interface and no guarantee is given on the values in the address space of the AnalyserChannel.
Maintenance	The AnalyserChannel is in the Maintenance mode. This mode is used to perform remote maintenance on the AnalyserChannel. To enter the Maintenance mode, the operator shall call the GotoMaintenance *Method* from the ADI *Client*. To return to the Operating mode, the operator shall call the GotoOperating *Method* from the ADI *Client*. When the AnalyserChannel is in the Maintenance mode, the parent AnalyserDevice has no control over it. In this mode, there is no guarantee given on the values in the address space.

Table 69 – AnalyserChannelStateMachineType States

BrowseName	References	Target BrowseName	Value	Target Type Definition	Notes
States					
SlaveMode	HasProperty	StateNumber	100	PropertyType	
	FromTransition	OperatingToSlaveModeTransition		TransitionType	
	FromTransition	MaintenanceToSlaveModeTransition		TransitionType	
	FromTransition	LocalToSlaveModeTransition		TransitionType	
	ToTransition	SlaveModeToOperatingTransition		TransitionType	
Operating	HasProperty	StateNumber	200	PropertyType	
	FromTransition	SlaveModeToOperatingTransition		TransitionType	
	FromTransition	MaintenanceToOperatingTransition		TransitionType	
	FromTransition	LocalToOperatingTransition		TransitionType	
	ToTransition	OperatingToLocalTransition		TransitionType	
	ToTransition	OperatingToMaintenanceTransition		TransitionType	
	ToTransition	OperatingToSlaveModeTransition		TransitionType	
Local	HasProperty	StateNumber	300	PropertyType	
	FromTransition	OperatingToLocalTransition		TransitionType	
	FromTransition	MaintenanceToLocalTransition		TransitionType	
	ToTransition	LocalToOperatingTransition		TransitionType	
	ToTransition	LocalToMaintenanceTransition		TransitionType	
	ToTransition	LocalToSlaveModeTransition		TransitionType	
Maintenance	HasProperty	StateNumber	400	PropertyType	
	FromTransition	OperatingToMaintenanceTransition		TransitionType	
	FromTransition	LocalToMaintenanceTransition		TransitionType	
	ToTransition	MaintenanceToOperatingTransition		TransitionType	
	ToTransition	MaintenanceToLocalTransition		TransitionType	
	ToTransition	MaintenanceToSlaveModeTransition		TransitionType	

The set of states defined to describe an *AnalyserChannel* can be expanded. Sub-states can be defined for the base states to provide more resolution to the process and to describe the cause and effects of additional stimuli and transitions.

5.3.3.4 AnalyserChannelStateMachineType Transitions

Transitions are instances of *Objects* of the *TransitionType* defined in [UA Part 5] – Appendix B which also includes the definitions of the FromState, ToState, HasCause, and HasEffect *References* used. Table 70 specifies the Transitions defined for the *AnalyserChannelStateMachineType.* Each Transition is assigned a unique *TransitionNumber*.

Table 70 – AnalyserChannelStateMachineType Transitions

BrowseName	References	Target BrowseName	Value	Target Type Definition	Notes
Transitions					
SlaveModeToOperatingTransition	HasProperty	TransitionNumber	1	PropertyType	
	FromState	SlaveMode		InitialStateType	
	ToState	Operating		AnalyserChannelOperatingStateType	
	HasCause				The AnalyserDevice moves from Local or Maintenance state to Operating state
OperatingToLocalTransition	HasProperty	TransitionNumber	2	PropertyType	
	FromState	Operating		AnalyserChannelOperatingStateType	
	ToState	Local		AnalyserChannelLocalStateType	
	HasCause	Press Local button on channel			External cause
OperatingToMaintenanceTransition	HasProperty	TransitionNumber	3	PropertyType	
	FromState	Operating		AnalyserChannelOperatingStateType	
	ToState	Maintenance		AnalyserChannelMaintenanceStateType	
	HasCause	GotoMaintenance		Method	
LocalToOperatingTransition	HasProperty	TransitionNumber	4	PropertyType	
	FromState	Local		AnalyserChannelLocalStateType	
	ToState	Operating		AnalyserChannelOperatingStateType	
	HasCause	Release Local button on channel			External cause
LocalToMaintenanceTransition	HasProperty	TransitionNumber	5	PropertyType	
	FromState	Local		AnalyserChannelLocalStateType	
	ToState	Maintenance		AnalyserChannelMaintenanceStateType	
	HasCause	Release Local button on channel			External cause
MaintenanceToOperatingTransition	HasProperty	TransitionNumber	6	PropertyType	
	FromState	Maintenance		AnalyserChannelMaintenanceStateType	
	ToState	Operating		AnalyserChannelOperatingStateType	
	HasCause	GotoOperating		Method	
MaintenanceToLocalTransition	HasProperty	TransitionNumber	7	PropertyType	
	FromState	Maintenance		AnalyserChannelMaintenanceStateType	
	ToState	Local		AnalyserChannelLocalStateType	

BrowseName	**References**	**Target BrowseName**	**Value**	**Target Type Definition**	**Notes**
	HasCause	Press Local button on channel			External cause
OperatingToSlaveModeTransition	HasProperty	TransitionNumber	8	PropertyType	
	FromState	Operating		AnalyserChannelOperatingStateType	
	ToState	SlaveMode		StateType	
	HasCause	AnalyserDevice moves from Operating to Local or Maintenance state.			External cause
LocalToSlaveModeTransition	HasProperty	TransitionNumber	9	PropertyType	
	FromState	Local		AnalyserChannelLocalStateType	
	ToState	SlaveMode		StateType	
	HasCause	AnalyserDevice moves from Operating to Local or Maintenance state.			External cause
MaintenanceToSlaveModeTransition	HasProperty	TransitionNumber	10	PropertyType	
	FromState	Maintenance		AnalyserChannelMaintenanceStateType	
	ToState	SlaveMode		StateType	
	HasCause	AnalyserDevice moves from Operating to Local or Maintenance state.			External cause

5.3.4 AnalyserChannel_OperatingModeSubStateMachineType

5.3.4.1 Introduction

AnalyserChannel_OperatingModeSubStateMachineType is a subtype of *FiniteStateMachineType*. The states are derived from the *ANSI/ISA TR 88.02-2008 Machine and Unit States* Technical Report [ISA-88 TR], which in turn were derived from the OMAC PackML tag definition set and the ANSI/ISA 88 Part 1 standard.

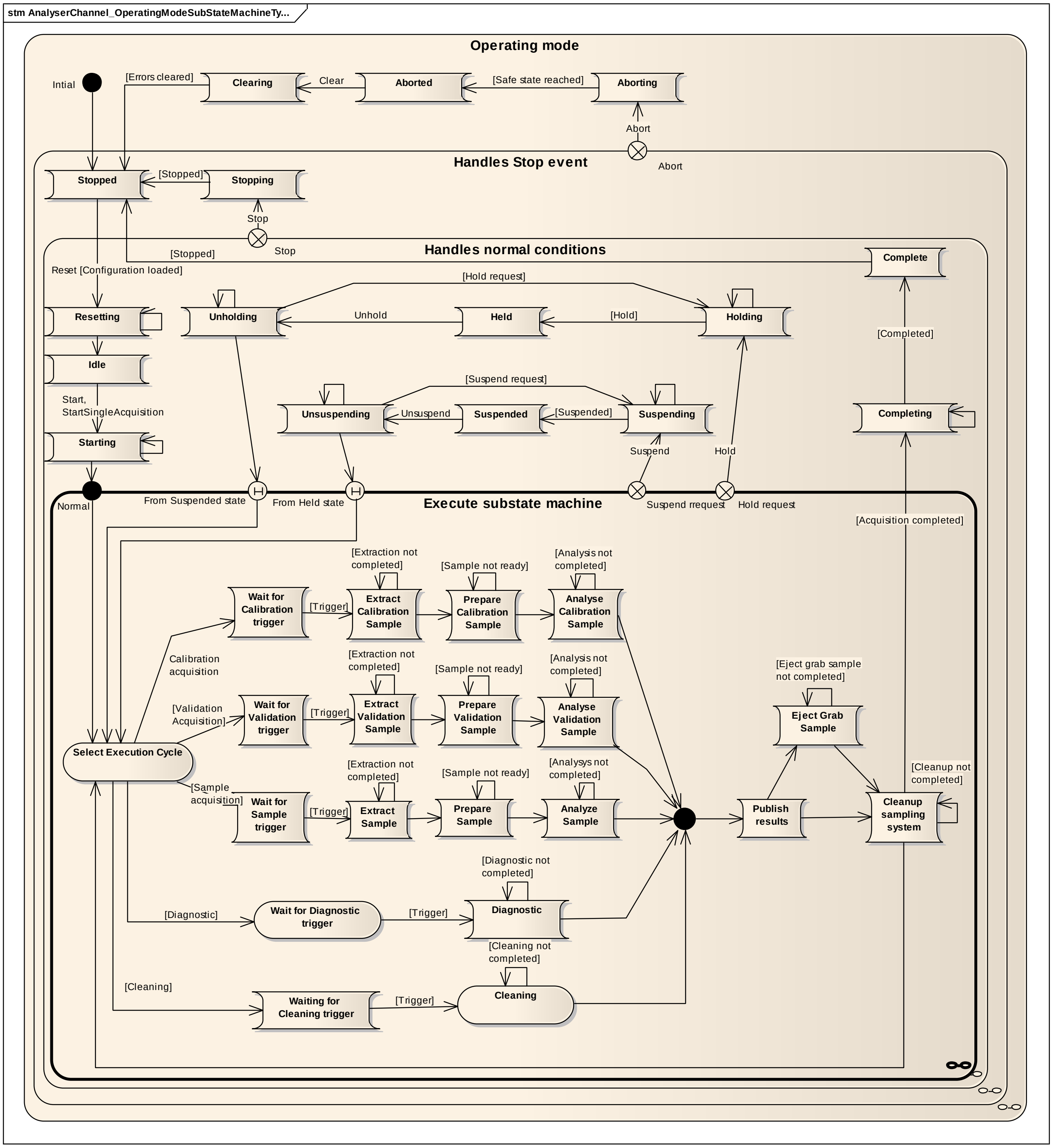

Figure 19 - AnalyserChannel_OperatingModeSubStateMachineType

When the *AnalyserChannel* is suspended or held:

- The normal Execute state is interrupted
- The actual Execute sub-state information shall be kept

When returning from Suspended or Held state:

- The restart point in Execute state shall be the junction point driven by the SelectExecutionCycle
- All sub-states shall be executed, but the vendor may use the information stored at the interruption point to optimize the execution of some sub-states.

5.3.4.2 Type definition: AnalyserChannel_OperatingModeSubStateMachineType *ObjectType*

The AnalyserChannel_OperatingModeSub*StateMachineType* is formally defined in Table 71.

Table 71 – AnalyserChannel_OperatingModeSubStateMachineType Definition

Attribute	**Value**				
BrowseName	AnalyserChannel_OperatingModeSubStateMachineType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**Data Type**	**Target Type Definition**	**Modelling Rule**
Subtype of the *FiniteStateMachineType* defined in [UA Part 5]					
HasComponent	Object	Stopped		InitialStateType	Mandatory
HasComponent	Object	Resetting		StateType	Mandatory
HasComponent	Object	Idle		StateType	Mandatory
HasComponent	Object	Starting		StateType	Mandatory
HasComponent	Object	Execute		AnalyserChannelOperatingModeExecuteStateType	Mandatory
HasComponent	Object	Completing		StateType	Mandatory
HasComponent	Object	Complete		StateType	Mandatory
HasComponent	Object	Suspending		StateType	Mandatory
HasComponent	Object	Suspended		StateType	Mandatory
HasComponent	Object	Unsuspending		StateType	Mandatory
HasComponent	Object	Holding		StateType	Mandatory
HasComponent	Object	Held		StateType	Mandatory
HasComponent	Object	Unholding		StateType	Mandatory
HasComponent	Object	Stopping		StateType	Mandatory
HasComponent	Object	Aborting		StateType	Mandatory
HasComponent	Object	Aborted		StateType	Mandatory
HasComponent	Object	Clearing		StateType	Mandatory
HasComponent	Object	StoppedToResettingTransition		TransitionType	Mandatory
HasComponent	Object	ResettingTransition		TransitionType	Mandatory
HasComponent	Object	ResettingToIdleTransition		TransitionType	Mandatory
HasComponent	Object	IdleToStartingTransition		TransitionType	Mandatory
HasComponent	Object	StartingTransition		TransitionType	Mandatory
HasComponent	Object	StartingToExecuteTransition		TransitionType	Mandatory
HasComponent	Object	ExecuteToCompletingTransition		TransitionType	Mandatory
HasComponent	Object	CompletingTransition		TransitionType	Mandatory
HasComponent	Object	CompletingToCompleteTransition		TransitionType	Mandatory
HasComponent	Object	CompleteToStoppedTransition		TransitionType	Mandatory
HasComponent	Object	ExecuteToHoldingTransition		TransitionType	Mandatory
HasComponent	Object	HoldingTransition		TransitionType	Mandatory
HasComponent	Object	HoldingToHeldTransition		TransitionType	Mandatory
HasComponent	Object	HeldToUnholdingTransition		TransitionType	Mandatory
HasComponent	Object	UnholdingTransition		TransitionType	Mandatory
HasComponent	Object	UnholdingToHoldingTransition		TransitionType	Mandatory
HasComponent	Object	UnholdingToExecuteTransition		TransitionType	Mandatory
HasComponent	Object	ExecuteToSuspendingTransition		TransitionType	Mandatory
HasComponent	Object	SuspendingTransition		TransitionType	Mandatory
HasComponent	Object	SuspendingToSuspendedTransition		TransitionType	Mandatory
HasComponent	Object	SuspendedToUnsuspendingTransition		TransitionType	Mandatory
HasComponent	Object	UnsuspendingTransition		TransitionType	Mandatory
HasComponent	Object	UnsuspendingToSuspendingTransition		TransitionType	Mandatory
HasComponent	Object	UnsuspendingToExecuteTransition		TransitionType	Mandatory
HasComponent	Object	StoppingToStoppedTransition		TransitionType	Mandatory
HasComponent	Object	AbortingToAbortedTransition		TransitionType	Mandatory

HasComponent	Object	AbortedToClearingTransition		TransitionType	Mandatory
HasComponent	Object	ClearingToStoppedTransition		TransitionType	Mandatory
HasComponent	Object	ResettingToStoppingTransition		TransitionType	Mandatory
HasComponent	Object	IdleToStoppingTransition		TransitionType	Mandatory
HasComponent	Object	StartingToStoppingTransition		TransitionType	Mandatory
HasComponent	Object	ExecuteToStoppingTransition		TransitionType	Mandatory
HasComponent	Object	CompletingToStoppingTransition		TransitionType	Mandatory
HasComponent	Object	CompleteToStoppingTransition		TransitionType	Mandatory
HasComponent	Object	SuspendingToStoppingTransition		TransitionType	Mandatory
HasComponent	Object	SuspendedToStoppingTransition		TransitionType	Mandatory
HasComponent	Object	UnsuspendingToStoppingTransition		TransitionType	Mandatory
HasComponent	Object	HoldingToStoppingTransition		TransitionType	Mandatory
HasComponent	Object	HeldToStoppingTransition		TransitionType	Mandatory
HasComponent	Object	UnholdingToStoppingTransition		TransitionType	Mandatory
HasComponent	Object	StoppedToAbortingTransition		TransitionType	Mandatory
HasComponent	Object	ResettingToAbortingTransition		TransitionType	Mandatory
HasComponent	Object	IdleToAbortingTransition		TransitionType	Mandatory
HasComponent	Object	StartingToAbortingTransition		TransitionType	Mandatory
HasComponent	Object	ExecuteToAbortingTransition		TransitionType	Mandatory
HasComponent	Object	CompletingToAbortingTransition		TransitionType	Mandatory
HasComponent	Object	CompleteToAbortingTransition		TransitionType	Mandatory
HasComponent	Object	SuspendingToAbortingTransition		TransitionType	Mandatory
HasComponent	Object	SuspendedToAbortingTransition		TransitionType	Mandatory
HasComponent	Object	UnsuspendingToAbortingTransition		TransitionType	Mandatory
HasComponent	Object	HoldingToAbortingTransition		TransitionType	Mandatory
HasComponent	Object	HeldToAbortingTransition		TransitionType	Mandatory
HasComponent	Object	UnholdingToAbortingTransition		TransitionType	Mandatory
HasComponent	Object	StoppingToAbortingTransition		TransitionType	Mandatory
HasComponent	Object	ExecuteSubStateMachine		AnalyserChannel_OperatingModeExecuteSubStateMachineType	Mandatory

Table 72 – AnalyserChannelOperatingModeExecuteStateType Definition

Attribute	Value				
BrowseName	AnalyserChannelOperatingModeExecuteStateType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**Data Type**	**TypeDefinition**	**Modelling Rule**
Subtype of the *StateType* defined in [UA Part 5]					
HasSubStateMachine	Object	ExecuteSubStateMachine		AnalyserChannel_OperatingModeExecuteSubStateMachineType	Mandatory

5.3.4.3 AnalyserChannel_OperatingModeSubStateMachineType States

Table 74 specifies the AnalyserChannel_OperatingModeSubStateMachineType*'s* State *Objects*. These State *Objects* are instances of the *StateType* defined in [UA Part 5] - Appendix B. Each State is assigned a unique *StateNumber* value. Subtypes of the AnalyserChannel_OperatingModeSubStateMachineType can add *Reference*s from any state to a subordinate or nested *StateMachine Object* to extend the *FiniteStateMachine*.

A standard set of states are defined for the AnalyserChannel_OperatingModeSub*StateMachineType*. These states represent the operational condition of the AnalyserChannel in Operating mode. All AnalyserChannels that contain an AnalyserChannel_OperatingModeSub*StateMachineType* must support this base set. A device may or may not require a *Client* action to cause the state to change. See Table 73 for the description of the states.

Table 73 – AnalyserChannel_OperatingModeSubStateMachineType State Descriptions

State No.	StateName	Description
1	Clearing	Initiated by Clear *Method* call, this state clears faults that may have occurred when Aborting and are present in the Aborted state before proceeding to a Stopped state. This state guarantees that the *Client* will see fault signals before going back to Stopped state.
2	Stopped	This is the initial state after AnalyserDeviceStateMachine state Powerup. At this point: • All communications with other systems are functioning (If applicable). • The state machine waits for a Reset or SetConfiguration *Method* call.
3	Starting	The analyser has received the Start or StartSingleAcquisition *Method* call and it is preparing to enter in Execute state. At this point: The analyser system shall be ready to start. Prepare the system for continuous acquisition. When completed, the state machine automatically goes in Execute state.
4	Idle	At the beginning of this state: • The Resetting state is completed • All *Parameters* have been committed • All analyser components are warmed-up and ready to start acquisition • Waiting for Start or StartSingleAcquisition *Method* call
5	Suspended	The analyser or channel may be running but no results are being generated while the analyser or channel is waiting for external process conditions to return to normal. When the offending process conditions return to normal, the Suspended state will transition to Unsuspending and hence continue towards the normal Execute state. At this state, no acquisition cycle is performed. Note: The Suspended state can be reached as a result of abnormal external process conditions and differs from Held in that Held is typically a result of an operator request or an automatically detected analyser or channel fault condition that should be corrected before an operator request to transition to the Unholding state will be processed.
6	Execute	All repetitive acquisition cycles are done in this state: • Wait for trigger • Grab sample from process • Prepare the sample for analysis • Analyse the sample • Publish results • Cleanup sampling system for next acquisition cycle See AnalyserChannel_*OperatingModeExecuteSubStateMachine* for more details.
7	Stopping	Initiated by a Stop *Method* call, this state: • Complete the ongoing acquisition if not too long • Get the actual acquisition (partial acquisition) • Discontinue the ongoing acquisition if partial acquisition does not make sense • Go to safe states gently, no rush Transitions automatically to Aborted state.

<table>
<tr><th>State No.</th><th>StateName</th><th>Description</th></tr>
<tr><td>8</td><td>Aborting</td><td>The Aborting state can be entered at any time in response to the Abort command or on the occurrence of a machine fault.
The aborting logic w ill bring the device to a rapid safe stop.
Operation of an Emergency Stop may cause the machine to be tripped by its safety system and may provide a signal to initiate the Aborting State.This state may include:
Abandoning the ongoing acquisition data
Rapidly putting the analyser system in safe states
Cooling dow n sampling cell
Closing cleaning solvent line
Closing sample inputs
Turning off Raman laser
Turning off source
All error conditions are saved and exposed in the AnalyserDevice/Channel.Status FunctionalGroup.
Transitions automatically to Aborted state.</td></tr>
<tr><td>9</td><td>Aborted</td><td>This state maintains machine status information relevant to the Abort condition.
The analyser is in safe state and:
Protects user and equipment
All error conditions are saved and exposed in the AnalyserDevice/Channel.Status FunctionalGroup.
The analyser can only exit the Aborted state after an explicit Clear Method call, often after manual intervention to correct and reset the detected device fault.</td></tr>
<tr><td>10</td><td>Holding</td><td>When the analyser or channel is in the Execute state, the Hold command can be used to start Holding logic w hich brings the analyser or channel to a controlled stop or to a state w hich represents Held for the particular unit control mode. An analyser or channel can go into this state either w hen an internal equipment fault is automatically detected or by an operator command. The Hold command offers the operator a safe way to intervene manually in the process (such as replacing solvent container) and restarting execution when conditions are safe.</td></tr>
<tr><td>11</td><td>Held</td><td>The Held state holds the analyser or channel's operation. At this state, no acquisition cycle is performed.</td></tr>
<tr><td>12</td><td>Unholding</td><td>The Unholding state is a response to an operator command to resume the Execute state.
Issuing the Unhold Method call w ill prepare the analyser or channel to re-enter the normal Execute state. The actions of this state may include:
Heating-up accessories
Reinitiating sampling system
Note that an operator Unhold command is alw ays required and Unholding can never be initiated automatically.</td></tr>
<tr><td>13</td><td>Suspending</td><td>This state is a result of a change in monitored conditions due to process conditions or factors. The trigger event w ill cause a temporary suspension of the Execute state. Suspending is typically the result of starvation of the process to analyse or or issues with the sampling system that prevents the analyser or channel from continued Execution. During the controlled sequence of Suspending the analyser or channel w ill transition to a Suspended state. The Suspending state might be forced by the operator using the Suspend Method call.</td></tr>
<tr><td>14</td><td>Unsuspending</td><td>This state is a result of a device request from Suspended state to transition back to the Execute state by calling the Unsuspend Method. The actions of this state may include:
Heating-up accessories
Reinitiating sampling system
This state is entered prior to the Execute state, and prepares the analyser or channel for the Execute state.</td></tr>
<tr><td>15</td><td>Resetting</td><td>This state is the result of a Reset or SetConfiguration Method call from the Stopped state. The Parameters are committed at this state. The actions of this state may include:
Resetting Hardw are
Analyser warm up
Enabling sampling sub-system
Enabling cleaning sampling path
Turning on source
Heating-up liquid cell
When completed, the state machine goes automatically to the Idle state.</td></tr>
</table>

State No.	StateName	Description
16	Completing	This state is an automatic or commanded exit from the Execute state. Normal operation has run to completion, i.e. the requested number of samples has been analysed. At this point, the pre-configured acquisition cycle(s) are completed. The actions of this state may include: • Flushing data path • Completing sample cells cleaning state • Going to safe states When done, it automatically transitions to the Complete state.
17	Complete	At this point, the Completing state is done and it transitions automatically to Stopped state to w ait. From an analyser point of view , this is almost a transient state.

Table 74 – AnalyserChannel_OperatingModeSubStateMachineType States

BrowseName	References	Target BrowseName	Value	Target Type Definition	Notes
States					
Stopped	HasProperty	StateNumber	2	PropertyType	
	FromTransition	CompleteToStoppedTransition		TransitionType	Method
	FromTransition	StoppingToStoppedTransition		TransitionType	Method
	FromTransition	ClearingToStoppedTransition		TransitionType	Method
	ToTransition	StoppedToResettingTransition		TransitionType	Method
	ToTransition	StoppedToAbortingTransition		TransitionType	Method
Resetting	HasProperty	StateNumber	15	PropertyType	
	FromTransition	StoppedToResettingTransition		TransitionType	Method
	ToTransition	ResettingToIdleTransition		TransitionType	Method
	ToTransition	ResettingToStoppingTransition		TransitionType	Method
	ToTransition	ResettingToAbortingTransition		TransitionType	Method
Idle	HasProperty	StateNumber	4	PropertyType	
	FromTransition	ResettingToIdleTransition		TransitionType	Method
	ToTransition	IdleToStartingTransition		TransitionType	Method
	ToTransition	idleToStoppingTransition		TransitionType	Method
	ToTransition	IdleToAbortingTransition		TransitionType	Method
Starting	HasProperty	StateNumber	3	PropertyType	
	FromTransition	IdleToStartingTransition		TransitionType	Method
	ToTransition	StartingToExecuteTransition		TransitionType	Method
	ToTransition	StartingToStoppingTransition		TransitionType	Method
	ToTransition	StartingToAbortingTransition		TransitionType	Method
Execute	HasProperty	StateNumber	6	PropertyType	
	FromTransition	StartingToExecuteTransition		TransitionType	Method
	ToTransition	ExecuteToCompletingTransition		TransitionType	Method
	ToTransition	ExecuteToStoppingTransition		TransitionType	Method
	ToTransition	ExecuteToAbortingTransition		TransitionType	Method

BrowseName	References	Target BrowseName	Value	Target Type Definition	Notes
Completing	HasProperty	StateNumber	16	PropertyType	
	FromTransition	ExecuteToCompletingTransition		TransitionType	Method
	ToTransition	CompletingToCompleteTransition		TransitionType	Method
	ToTransition	CompletingToStoppingTransition		TransitionType	Method
	ToTransition	CompletingToAbortingTransition		TransitionType	Method
Complete	HasProperty	StateNumber	17	PropertyType	
	FromTransition	CompletingToCompleteTransition		TransitionType	Method
	ToTransition	CompleteToStoppedTransition		TransitionType	Method
	ToTransition	CompleteToStoppingTransition		TransitionType	Method
	ToTransition	CompleteToAbortingTransition		TransitionType	Method
Suspending	HasProperty	StateNumber	13	PropertyType	
	FromTransition	ExecuteToSuspendingTransition		TransitionType	Method
	ToTransition	SuspendingToSuspendedTransition		TransitionType	Method
	ToTransition	SuspendingToStoppingTransition		TransitionType	Method
	ToTransition	SuspendingToAbortingTransition		TransitionType	Method
Suspended	HasProperty	StateNumber	5	PropertyType	
	FromTransition	SuspendingToSuspendedTransition		TransitionType	Method
	ToTransition	SuspendedToUnsuspendingTransition		TransitionType	Method
	ToTransition	SuspendedToStoppingTransition		TransitionType	Method
	ToTransition	SuspendiedToAbortingTransition		TransitionType	Method
Unsuspending	HasProperty	StateNumber	14	PropertyType	
	FromTransition	SuspendedToUnsuppendingTransition		TransitionType	Method
	ToTransition	UnsuppendingToExecuteTransition		TransitionType	Method
	ToTransition	UnsuppendingToSuspendingTransition		TransitionType	Method
	ToTransition	UnsuppendingToStoppingTransition		TransitionType	Method
	ToTransition	UnsuppendingToAbortingTransition		TransitionType	Method
Holding	HasProperty	StateNumber	10	PropertyType	
	FromTransition	ExecuteToHoldingTransition		TransitionType	Method
	ToTransition	HoldingToHeldTransition		TransitionType	Method
	ToTransition	HoldingToStoppingTransition		TransitionType	Method
	ToTransition	HoldingToAbortingTransition		TransitionType	Method
Held	HasProperty	StateNumber	11	PropertyType	
	FromTransition	HoldingToHeldTransition		TransitionType	Method
	ToTransition	HeldToUnholdingTransition		TransitionType	Method
	ToTransition	HeldToStoppingTransition		TransitionType	Method
	ToTransition	HeldToAbortingTransition		TransitionType	Method
Unholding	HasProperty	StateNumber	12	PropertyType	
	FromTransition	HeldToUnholdingTransition		TransitionType	Method
	ToTransition	UnholdingToExecuteTransition		TransitionType	Method
	ToTransition	UnholdingToHoldingTransition		TransitionType	Method
	ToTransition	UnholdingToStoppingTransition		TransitionType	Method
	ToTransition	UnholdingToAbortingTransition		TransitionType	Method

BrowseName	References	Target BrowseName	Value	Target Type Definition	Notes
Stopping	HasProperty	StateNumber	7	PropertyType	
	FromTransition	ResettingToStoppingTransition		TransitionType	Method
	FromTransition	IdleToStoppingTransition		TransitionType	Method
	FromTransition	StartingToStoppingTransition		TransitionType	Method
	FromTransition	ExecuteToStoppingTransition		TransitionType	Method
	FromTransition	CompletingToStoppingTransition		TransitionType	Method
	FromTransition	CompleteToStoppingTransition		TransitionType	Method
	FromTransition	SuspendingToStoppingTransition		TransitionType	Method
	FromTransition	SuspendedToStoppingTransition		TransitionType	Method
	FromTransition	UnsuspendingToStoppingTransition		TransitionType	Method
	FromTransition	HoldingToStoppingTransition		TransitionType	Method
	FromTransition	HeldToStoppingTransition		TransitionType	Method
	ToTransition	StoppingToStoppedTransition		TransitionType	Method
	ToTransition	StoppingToAbortingTransition		TransitionType	Method
Aborting	HasProperty	StateNumber	8	PropertyType	
	FromTransition	StoppingToAbortingTransition		TransitionType	Method
	FromTransition	StoppedToAbortingTransition		TransitionType	Method
	FromTransition	ResettingToAbortingTransition		TransitionType	Method
	FromTransition	IdleToAbortingTransition		TransitionType	Method
	FromTransition	StartingToAbortingTransition		TransitionType	Method
	FromTransition	ExecuteToAbortingTransition		TransitionType	Method
	FromTransition	CompletingToAbortingTransition		TransitionType	Method
	FromTransition	CompleteToAbortingTransition		TransitionType	Method
	FromTransition	SuspendingToAbortingTransition		TransitionType	Method
	FromTransition	SuspendedToAbortingTransition		TransitionType	Method
	FromTransition	UnsuspendingToAbortingTransition		TransitionType	Method
	FromTransition	HoldingToAbortingTransition		TransitionType	Method
	FromTransition	HelpToAbortingTransition		TransitionType	Method
	FromTransition	UnholdingToAbortingTransition		TransitionType	Method
	ToTransition	AbortingToAbortedTransition		TransitionType	Method
Aborted	HasProperty	StateNumber	9	PropertyType	
	FromTransition	AbortingToAbortedTransition		TransitionType	Method
	ToTransition	AbortedToClearingTransition		TransitionType	Method
Clearing	HasProperty	StateNumber	1	PropertyType	
	FromTransition	AbortedToClearingTransition		TransitionType	Method
	ToTransition	ClearingToStoppedTransition		TransitionType	Method

The set of states defined to describe in AnalyserChannel_OperatingModeSub*StateMachineType* can be expanded. Sub-states can be defined for the base states to provide more resolution to the process and to describe the cause and effects of additional stimuli and transitions. For example, the “Stopped” state can include the sub states “Preparing” and “Done” to indicate if the function is still preparing the device or if it has completed preparation

5.3.4.4 AnalyserChannel_OperatingModeSubStateMachineType Transitions

Transitions are instances of *Objects* of the *TransitionType* defined in [UA Part 5] - State Machine Appendix which also includes the definitions of the ToState, FromState, HasCause, and HasEffect *Reference*s used. Table 75 specifies the Transitions defined for the

AnalyserChannel_OperatingModeSub*StateMachineType.* Each Transition is assigned a unique *TransitionNumber*.

Table 75 – AnalyserChannel_OperatingModeSubStateMachine Transitions

BrowseName	References	Target BrowseName	Value	Target Type Definition	Notes
Transitions					
StoppedToResettingTransition	HasProperty	TransitionNumber	1	PropertyType	
	FromState	Stopped		StateType	
	ToState	Resetting		StateType	
	HasCause	Reset		Method	
	HasCause	SetConfiguration		Method	
ResettingTransition	HasProperty	TransitionNumber	2	PropertyType	
	FromState	Resetting		StateType	
	ToState	Resetting		StateType	
ResettingToIdleTransition	HasProperty	TransitionNumber	3	PropertyType	
	FromState	Resetting		StateType	
	ToState	Idle		StateType	
IdleToStartingTransition	HasProperty	TransitionNumber	4	PropertyType	
	FromState	Idle		StateType	
	ToState	Starting		StateType	
	HasCause	Start		Method	
	HasCause	StartSingleAcquisition		Method	
StartingTransition	HasProperty	TransitionNumber	5	PropertyType	
	FromState	Starting		StateType	
	ToState	Starting		StateType	
StartingToExecuteTransition	HasProperty	TransitionNumber	6	PropertyType	
	FromState	Starting		StateType	
	ToState	Execute		StateType	
ExecuteToCompletingTransition	HasProperty	TransitionNumber	7	PropertyType	
	FromState	Execute		StateType	
	ToState	Completing		StateType	
CompletingTransition	HasProperty	TransitionNumber	8	PropertyType	
	FromState	Completing		StateType	
	ToState	Completing		StateType	
CompletingToCompleteTransition	HasProperty	TransitionNumber	9	PropertyType	
	FromState	Completing		StateType	
	ToState	Complete		StateType	
CompleteToStoppedTransition	HasProperty	TransitionNumber	10	PropertyType	
	FromState	Complete		StateType	
	ToState	Stopped		StateType	
ExecuteToHoldingTransition	HasProperty	TransitionNumber	11	PropertyType	
	FromState	Execute		StateType	
	ToState	Holding		StateType	
	HasCause	Hold		Method	
HoldingTransition	HasProperty	TransitionNumber	12	PropertyType	
	FromState	Holding		StateType	
	ToState	Holding		StateType	
HoldingToHeldTransition	HasProperty	TransitionNumber	13	PropertyType	
	FromState	Holding		StateType	
	ToState	Held		StateType	

BrowseName	References	Target BrowseName	Value	Target Type Definition	Notes
HeldToUnholdingTransition	HasProperty	TransitionNumber	14	PropertyType	
	FromState	Held		StateType	
	ToState	Unholding		StateType	
	HasCause	Unhold		Method	
UnholdingTransition	HasProperty	TransitionNumber	15	PropertyType	
	FromState	Unholding		StateType	
	ToState	Unholding		StateType	
UnholdingToHoldingTransition	HasProperty	TransitionNumber	16	PropertyType	
	FromState	Unholding		StateType	
	ToState	Holding		StateType	
	HasCause	Hold		Method	
UnholdingToExecuteTransition	HasProperty	TransitionNumber	17	PropertyType	
	FromState	Unholding		StateType	
	ToState	Execute		StateType	
ExecuteToSuspendingTransition	HasProperty	TransitionNumber	18	PropertyType	
	FromState	Execute		StateType	
	ToState	Suspending		StateType	
	HasCause	Suspend		Method	
SuspendingTransition	HasProperty	TransitionNumber	19	PropertyType	
	FromState	Suspending		StateType	
	ToState	Suspending		StateType	
SuspendingToSuspendedTransition	HasProperty	TransitionNumber	20	PropertyType	
	FromState	Suspending		StateType	
	ToState	Suspended		StateType	
SuspendedToUnsuspendingTransition	HasProperty	TransitionNumber	21	PropertyType	
	FromState	Suspended		StateType	
	ToState	Unsuspending		StateType	
	HasCause	Unsuspend		Method	
UnsuspendingTransition	HasProperty	TransitionNumber	22	PropertyType	
	FromState	Unsuspending		StateType	
	ToState	Unsuspending		StateType	
UnsuspendingToSuspendingTransition	HasProperty	TransitionNumber	23	PropertyType	
	FromState	Unsuspending		StateType	
	ToState	Suspending		StateType	
	HasCause	Suspend		Method	
UnsuspendingToExecuteTransition	HasProperty	TransitionNumber	24	PropertyType	
	FromState	Unsuspending		StateType	
	ToState	Execute		StateType	
StoppingToStoppedTransition	HasProperty	TransitionNumber	25	PropertyType	
	FromState	Stopping		StateType	
	ToState	Stopped		StateType	
AbortingToAbortedTransition	HasProperty	TransitionNumber	26	PropertyType	
	FromState	Aborting		StateType	
	ToState	Aborted		StateType	

BrowseName	References	Target BrowseName	Value	Target Type Definition	Notes
AbortedToClearingTransition	HasProperty	TransitionNumber	27	PropertyType	
	FromState	Aborted		StateType	
	ToState	Clearing		StateType	
	HasCause	Clear		Method	
ClearingToStoppedTransition	HasProperty	TransitionNumber	28	PropertyType	
	FromState	Clearing		StateType	
	ToState	Stopped		StateType	
ResettingToStoppingTransition	HasProperty	TransitionNumber	29	PropertyType	
	FromState	Resetting		StateType	
	ToState	Stopping		StateType	
	HasCause	Stop		Method	
IdleToStoppingTransition	HasProperty	TransitionNumber	30	PropertyType	
	FromState	Idle		StateType	
	ToState	Stopping		StateType	
	HasCause	Stop		Method	
StartingToStoppingTransition	HasProperty	TransitionNumber	31	PropertyType	
	FromState	Starting		StateType	
	ToState	Stopping		StateType	
	HasCause	Stop		Method	
ExecuteToStoppingTransition	HasProperty	TransitionNumber	32	PropertyType	
	FromState	Execute		StateType	
	ToState	Stopping		StateType	
	HasCause	Stop		Method	
CompletingToStoppingTransition	HasProperty	TransitionNumber	33	PropertyType	
	FromState	Completing		StateType	
	ToState	Stopping		StateType	
	HasCause	Stop		Method	
CompleteToStoppingTransition	HasProperty	TransitionNumber	34	PropertyType	
	FromState	Complete		StateType	
	ToState	Stopping		StateType	
	HasCause	Stop		Method	
SuspendingToStoppingTransition	HasProperty	TransitionNumber	35	PropertyType	
	FromState	Suspending		StateType	
	ToState	Stopping		StateType	
	HasCause	Stop		Method	
SuspendedToStoppingTransition	HasProperty	TransitionNumber	36	PropertyType	
	FromState	Suspended		StateType	
	ToState	Stopping		StateType	
	HasCause	Stop		Method	
UnsuspendingToStoppingTransition	HasProperty	TransitionNumber	37	PropertyType	
	FromState	Unsuspending		StateType	
	ToState	Stopping		StateType	
	HasCause	Stop		Method	
HoldingToStoppingTransition	HasProperty	TransitionNumber	38	PropertyType	
	FromState	Holding		StateType	

BrowseName	**References**	**Target BrowseName**	**Value**	**Target Type Definition**	**Notes**
	ToState	Stopping		StateType	
	HasCause	Stop		Method	
HeldToStoppingTransition	HasProperty	TransitionNumber	39	PropertyType	
	FromState	Held		StateType	
	ToState	Stopping		StateType	
	HasCause	Stop		Method	
UnholdingToStoppingTransition	HasProperty	TransitionNumber	40	PropertyType	
	FromState	Unholding		StateType	
	ToState	Stopping		StateType	
	HasCause	Stop		Method	
StoppedToAbortingTransition	HasProperty	TransitionNumber	41	PropertyType	
	FromState	Stopped		StateType	
	ToState	Aborting		StateType	
	HasCause	Abort		Method	
ResettingToAbortingTransition	HasProperty	TransitionNumber	42	PropertyType	
	FromState	Resetting		StateType	
	ToState	Aborting		StateType	
	HasCause	Abort		Method	
IdleToAbortingTransition	HasProperty	TransitionNumber	43	PropertyType	
	FromState	Idle		StateType	
	ToState	Aborting		StateType	
	HasCause	Abort		Method	
StartingToAbortingTransition	HasProperty	TransitionNumber	44	PropertyType	
	FromState	Starting		StateType	
	ToState	Aborting		StateType	
	HasCause	Abort		Method	
ExecuteToAbortingTransition	HasProperty	TransitionNumber	45	PropertyType	
	FromState	Execute		StateType	
	ToState	Aborting		StateType	
	HasCause	Abort		Method	
CompletingToAbortingTransition	HasProperty	TransitionNumber	46	PropertyType	
	FromState	Completing		StateType	
	ToState	Aborting		StateType	
	HasCause	Abort		Method	
CompleteToAbortingTransition	HasProperty	TransitionNumber	47	PropertyType	
	FromState	Complete		StateType	
	ToState	Aborting		StateType	
	HasCause	Abort		Method	
SuspendingToAbortingTransition	HasProperty	TransitionNumber	48	PropertyType	
	FromState	Suspending		StateType	
	ToState	Aborting		StateType	
	HasCause	Abort		Method	
SuspendedToAbortingTransition	HasProperty	TransitionNumber	49	PropertyType	
	FromState	Suspended		StateType	
	ToState	Aborting		StateType	
	HasCause	Abort		Method	

BrowseName	References	Target BrowseName	Value	Target Type Definition	Notes
UnsuspendingToAbortingTransition	HasProperty	TransitionNumber	50	PropertyType	
	FromState	Unsuspending		StateType	
	ToState	Aborting		StateType	
	HasCause	Abort		Method	
HoldingToAbortingTransition	HasProperty	TransitionNumber	51	PropertyType	
	FromState	Holding		StateType	
	ToState	Aborting		StateType	
	HasCause	Abort		Method	
HeldToAbortingTransition	HasProperty	TransitionNumber	52	PropertyType	
	FromState	Held		StateType	
	ToState	Aborting		StateType	
	HasCause	Abort		Method	
UnholdingToAbortingTransition	HasProperty	TransitionNumber	53	PropertyType	
	FromState	Unholding		StateType	
	ToState	Aborting		StateType	
	HasCause	Abort		Method	
StoppingToAbortingTransition	HasProperty	TransitionNumber	54	PropertyType	
	FromState	Stopping		StateType	
	ToState	Aborting		StateType	
	HasCause	Abort		Method	

The *Reset* transition specifies the Transition from the *Complete* or *Stopped* to the *Resetting* State. It may be caused by the *Reset Method* or by the *SetConfiguration Method.*

The *Start* transition specifies the Transition from the *Idle* to the *Starting* State. It may be caused by the *Start Method.*

The *Stop* transition specifies the Transition from the *Stopping, Idle, Resetting, Unholding, Starting, Unsuspending, Held, Execute, Suspend, Holding, Completing, Suspending,* or *Complete* to the *Stopping* State. It may be caused by the *Stop Method.*

The *Hold* transition specifies the Transition from the *Unholding* or *Execute* to the *Holding* State. It may be caused by the *Hold Method.*

The *Unhold* transition specifies the Transition from the *Held* to the *Unholding* State. It may be caused by the *Unhold Method.*

The *Suspend* transition specifies the Transition from the *Unsuspending* or *Execute* to the *Suspending* State. It may be caused by the *Suspend Method.*

The *Abort* transition specifies the Transition from the *Stopping, Idle, Resetting, Unholding, Starting, Unsuspending, Held, Execute, Suspend, Holding, Completing, Suspending, Complete, Clearing, Stopped,* or *Stopping* to the *Aborting* State. It may be caused by the *Abort Method.*

The *Clear* transition specifies the Transition from the *Aborted* to the *Clearing* State. It may be caused by the *Clear Method.*

The *Complete* transition specifies the Transition from the *Execute* to the *Completing* State.

5.3.4.5 AnalyserChannel_OperatingModeExecuteSubStateMachineType

5.3.4.5.1 Introduction

The AnalyserChannel_*OperatingModeExecuteSubStateMachineType* describes the sub-states of the *AnalyserChannel_OperatingModeStateMachine* state *Execute*. Figure 20 illustrates components of AnalyserChannel_*OperatingModeExecuteSubStateMachineType.*

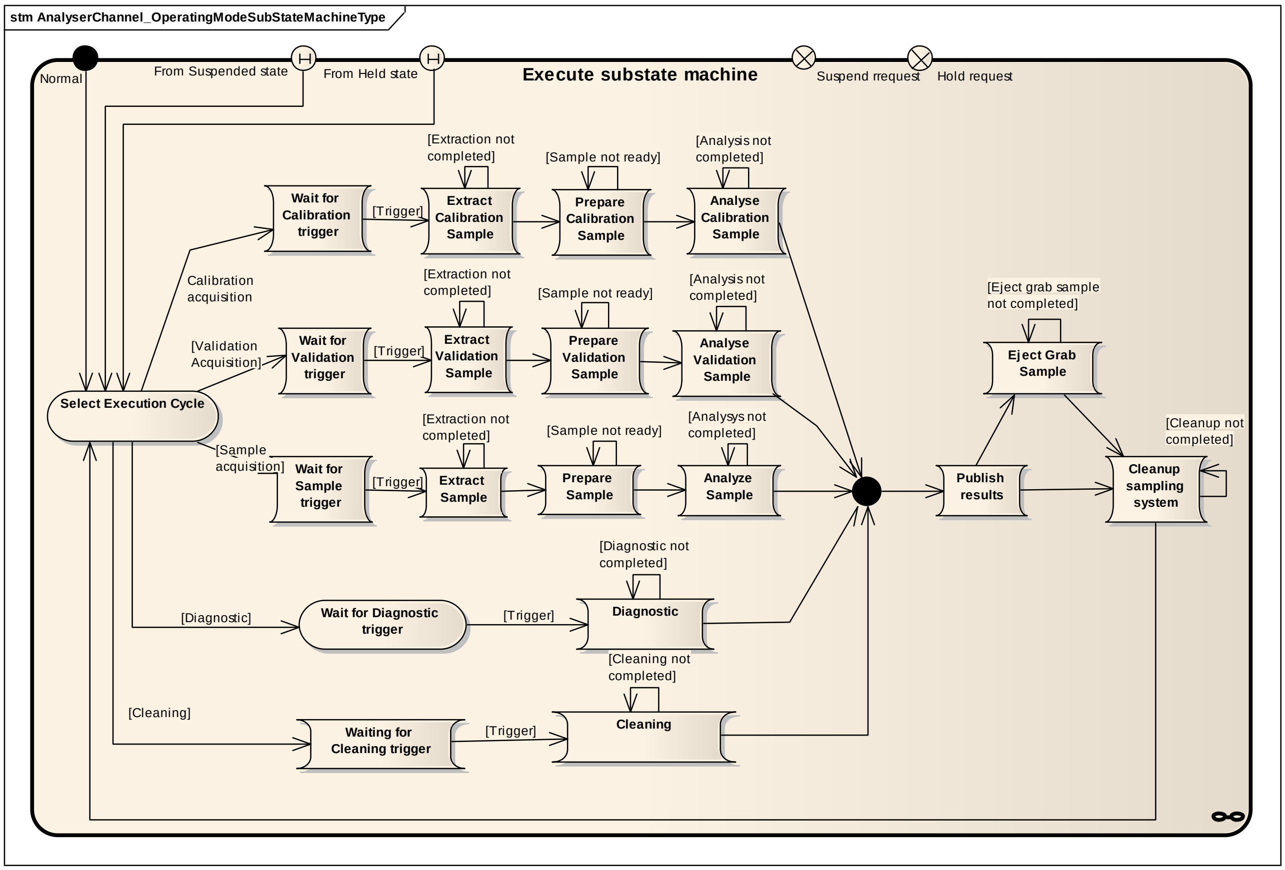

Figure 20 - AnalyserChannel_OperatingModeExecuteSubStateMachineType

5.3.4.5.2 Type definition: AnalyserChannel_*OperatingModeExecuteSub*StateMachineType *ObjectType*

AnalyserChannel_*OperatingModeExecuteSubStateMachineType* is formally defined in Table 76.

Table 76 – AnalyserChannel_*OperatingModeExecuteSub*StateMachineType Definition

Attribute	**Value**				
BrowseName	AnalyserChannel_*OperatingModeExecuteSub*StateMachineType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**Data Type**	**Type Definition**	**Modelling Rule**
Subtype of the *FiniteStateMachineType* defined in [UA Part 5]					
HasComponent	Object	SelectExecutionCycle		InitialStateType	Mandatory
HasComponent	Object	WaitForCalibrationTrigger		StateType	Mandatory
HasComponent	Object	ExtractCalibrationSample		StateType	Mandatory
HasComponent	Object	PrepareCalibrationSample		StateType	Mandatory
HasComponent	Object	AnalyseCalibrationSample		StateType	Mandatory
HasComponent	Object	WaitForValidationTrigger		StateType	Mandatory
HasComponent	Object	ExtractValidationSample		StateType	Mandatory
HasComponent	Object	PrepareValidationSample		StateType	Mandatory
HasComponent	Object	AnalyseValidationSample		StateType	Mandatory
HasComponent	Object	WaitForSampleTrigger		StateType	Mandatory
HasComponent	Object	ExtractSample		StateType	Mandatory
HasComponent	Object	PrepareSample		StateType	Mandatory
HasComponent	Object	AnalyseSample		StateType	Mandatory
HasComponent	Object	WaitForDiagnosticTrigger		StateType	Mandatory
HasComponent	Object	Diagnostic		StateType	Mandatory
HasComponent	Object	WaitForCleaningTrigger		StateType	Mandatory
HasComponent	Object	Cleaning		StateType	Mandatory
HasComponent	Object	PublishResults		StateType	Mandatory
HasComponent	Object	EjectGrabSample		StateType	Mandatory
HasComponent	Object	CleanupSamplingSystem		StateType	Mandatory
HasComponent	Object	SelectExecutionCycleToWaitForCalibrationTriggerTransition		TransitionType	Mandatory
HasComponent	Object	WaitForCalibrationTriggerToExtractCalibrationSampleTransition		TransitionType	Mandatory
HasComponent	Object	ExtractCalibrationSampleTransition		TransitionType	Mandatory
HasComponent	Object	ExtractCalibrationSampleToPrepareCalibrationSampleTransition		TransitionType	Mandatory
HasComponent	Object	PrepareCalibrationSampleTransition		TransitionType	Mandatory
HasComponent	Object	PrepareCalibrationSampleToAnalyseCalibrationSampleTransition		TransitionType	Mandatory
HasComponent	Object	AnalyseCalibrationSampleTransition		TransitionType	Mandatory
HasComponent	Object	AnalyseCalibrationSampleToPublishResultsTransition		TransitionType	Mandatory
HasComponent	Object	SelectExecutionCycleToWaitForTriggerValidationTransition		TransitionType	Mandatory
HasComponent	Object	WaitForValidationTriggerToExtractValidationSampleTransition		TransitionType	Mandatory
HasComponent	Object	ExtractValidationSampleTransition		TransitionType	Mandatory
HasComponent	Object	ExtractValidationSampleToPrepareValidationSampleTransition		TransitionType	Mandatory
HasComponent	Object	PrepareValidationSampleTransition		TransitionType	Mandatory
HasComponent	Object	PrepareValidationSampleToAnalyseValidationSampleTransition		TransitionType	Mandatory
HasComponent	Object	AnalyseValidationSampleTransition		TransitionType	Mandatory
HasComponent	Object	AnalyseValidationSampleToPublishResultsTransition		TransitionType	Mandatory
HasComponent	Object	SelectExecutionCycleToWaitForSampleTriggerTransition		TransitionType	Mandatory
HasComponent	Object	WaitForSampleTriggerToExtractSampleTransition		TransitionType	Mandatory
HasComponent	Object	ExtractSampleTransition		TransitionType	Mandatory
HasComponent	Object	ExtractSampleToPrepareSampleTransition		TransitionType	Mandatory
HasComponent	Object	PrepareSampleTransition		TransitionType	Mandatory
HasComponent	Object	PrepareSampleToAnalyseSampleTransition		TransitionType	Mandatory
HasComponent	Object	AnalyseSampleTransition		TransitionType	Mandatory
HasComponent	Object	AnalyseSampleToPublishResultsTransition		TransitionType	Mandatory
HasComponent	Object	SelectExecutionCycleToWaitForDiagnosticTriggerTransition		TransitionType	Mandatory
HasComponent	Object	WaitForDiagnosticTriggerToDiagnosticTransition		TransitionType	Mandatory
HasComponent	Object	DiagnosticTransition		TransitionType	Mandatory

HasComponent	Object	DiagnosticToPublishResultsTransition		TransitionType	Mandatory
HasComponent	Object	SelectExecutionCycleToWaitForCleaningTriggerTransition		TransitionType	Mandatory
HasComponent	Object	WaitForCleaningTriggerToCleaningTransition		TransitionType	Mandatory
HasComponent	Object	CleaningTransition		TransitionType	Mandatory
HasComponent	Object	CleaningToPublishResultsTransition		TransitionType	Mandatory
HasComponent	Object	PublishResultsToCleanupSamplingSystemTransition		TransitionType	Mandatory
HasComponent	Object	PublishResultsToEjectGrabSampleTransition		TransitionType	Mandatory
HasComponent	Object	EjectGrabSampleTransition		TransitionType	Mandatory
HasComponent	Object	EjectGrabSampleToCleanupSamplingSystemTransition		TransitionType	Mandatory
HasComponent	Object	CleanupSamplingSystemTransition		TransitionType	Mandatory
HasComponent	Object	CleanupSamplingSystemToSelectExecutionCycleTransition		TransitionType	Mandatory

5.3.4.5.3 AnalyserChannel_*OperatingModeExecuteSub*StateMachineType States

Table 78 specifies the AnalyserChannel_*OperatingModeExecuteSubStateMachine's* State *Objects*. These State *Objects* are instances of the *StateType* defined in [UA Part 5] – Appendix B. Each State is assigned a unique *StateNumber* value. Subtypes of the AnalyserChannel_*OperatingModeExecuteSubStateMachineType* can add *Reference*s from any state to a subordinate or nested *StateMachine Object* to extend the *FiniteStateMachine*.

A standard set of sub-states are defined for *AnalyseChannel_OperatingModeExecuteSubStateMachineType*. These sub-states represent the operational condition of the *AnalyseChannel_OperatingModeSubStateMachine Execute* state. All *the sub-states* must be supported, though they can be transient states.

Table 77 – AnalyserChannel_*OperatingModeExecuteSub*StateMachineType State Descriptions

StateName	Description
SelectExecutionCycle	This pseudo-state is used to decide w hich acquisition path shall be taken. This decision is made using a *Parameter* ExecutionCycle that can be: • Provided as a *Parameter* of the Start*SingleAcquisitiont Method* • Update by the vendor specific software and exposed in the *AnalyserChannel.AcquisitionStatus FunctionalGroup* The state machine w aits at this state until the underlying system is ready to take a given acquisition path.
WaitForCalibrationTrigger	Wait until the analyser channel is ready to perform the Calibration acquisition cycle, for example: • The external trigger is received from another system • A vendor specific *Parameter* in the *AcquisitionSettings* has been updated For analysers that do not need the step, the state is transient.
ExtractCalibrationSample	Collect / setup the sampling system to perform the acquisition cycle of a Calibration cycle, for example: • Empty and dry the sample liquid cell. • Place a calibrated sample in the acquisition path. For analysers that do not need the step, the state is transient.
PrepareCalibrationSample	Prepare the Calibration sample for the AnalyseCalibrationSample state ,for example: • Heating the Calibration sample • Homogenizing the Calibration sample For analysers that do not need the step, the state is transient.
AnalyseCalibrationSample	Perform the analysis of the Calibration Sample, for example: • Collect the reference spectrum • Collect the particle size histogram
WaitForValidationTrigger	Wait until the analyser channel is ready to perform the Validation acquisition cycle, for example: • The external trigger is received from another system • A vendor specific *Parameter* in the *AcquisitionSettings* has been updated For analysers that do not need the step, the state is transient.
ExtractValidationSample	Collect / setup the sampling system to perform the acquisition cycle of a Validation cycle, for example: • Empty and dry the sample liquid cell. • Place a calibrated sample in the acquisition path. For analysers that do not need the step, the state is transient.
PrepareValidationSample	Prepare the Validation sample for the AnalyseValidationSample state ,for example: • Heating the Validation sample • Homogenizing the Validation sample For analysers that do not need the step, the state is transient.
AnalyseValidationSample	Perform the analysis of the Validation Sample, for example: • Collect the Validation spectrum and compare it w ith the expected values
WaitForSampleTrigger	Wait until the analyser channel is ready to perform the Sample acquisition cycle, for example: • The external trigger is received from another system, like an external sampling system • A vendor specific *Parameter* in the *AcquisitionSettings* has been updated For analysers that do not need the step, the state is transient.
ExtractSample	Collect the Sample from the process, for example: • Physically extract a Sample from the process to fill a liquid cell • Extract pow der from the blender Some analyser probes do not need to extract the Sample from the process, for example a NIR reflectance probe. In this case, this state is a pass-through.
PrepareSample	Prepare the Sample for the AnalyseSample state, for example: • Heating the Sample • Homogenizing the Sample For analysers that do not need the step, the state is transient.
AnalyseSample	Perform the analysis of the Sample, for example: • Collect the Sample spectrum • Collect the Sample particle size histogram • Collect the Sample chromatogram
WaitForDiagnosticTrigger	Wait until the analyser channel is ready to perform the Diagnostic cycle, for example: • The external trigger is received from another system • A vendor specific *Parameter* in the *AcquisitionSettings* has been updated For analysers that do not need the step, the state is transient.

StateName	Description
Diagnostic	Perform the Diagnostic cycle. This cycle is a placeholder allowing the analyser vendor to extend this state to represent vendor specific analyser diagnostic cycles.
WaitForCleaningTrigger	Wait until the analyser channel is ready to perform the cleaning acquisition cycle, for example: • The external trigger is received from another system • A vendor specific *Parameter* in the *AcquisitionSettings* has been updated For analysers that do not need the step, the state is transient.
Cleaning	Perform the cleaning cycle.
PublishResults	Publish the results of the previous acquisition cycle. When the transition from *PublishResults* to *CleanupSamplingSystem* occurs, all results must be available.
EjectGrabSample	The Sample that was just analysed is ejected from the system to allow the operator or another system to grab it and send it to a control lab for example.
CleanupSamplingSystem	Cleanup the sampling sub-system to be ready for the next acquisition, for example: • Flush the liquid cell with a solvent For in-process probes, this state is transient.

The set of states defined to describe an AnalyserChannel_*OperatingModeExecuteSubStateMachine* can be expanded. Sub-states can be defined for the base states to provide more resolution to the process and to describe the cause and effects of additional stimuli and transitions. See Table 77 for a description of the states.

ExecutionCycle, ExecutionCycleSubcode and ActiveStream *Parameters* are set during the SelectExecutionCycle state. From the end of SelectExecutionCycle to the end of CleanupSamplingSystem, these two *Parameters* shall not change.

ExecutionCycle, ExecutionCycleSubcode, ActiveStream and IsActive *Parameters* are set during the SelectExecutionCycle state. From the end of SelectExecutionCycle to the end of CleanupSamplingSystem, these two *Parameters* shall not change.

WaitForxxxTrigger states represent waiting for situation like:

- External input i/o visible or not in the address space
- Internal timer (visible or not in the address space)

Table 78 – AnalyserChannel_*OperatingModeExecuteSub*StateMachineType States

BrowseName	References	Target BrowseName	Value	Target Type Definition
States				
SelectExecutionCycle	HasProperty	StateNumber	100	PropertyType
	FromTransition	CleanupSamplingSystemToSelectExecutionCycleTransition		TransitionType
	ToTransition	SelectExecutionCycleToWaitForCalibrationTriggerTransition		TransitionType
	ToTransition	SelectExecutionCycleToWaitForValidationTriggerTransition		TransitionType
	ToTransition	SelectExecutionCycleToWaitForSampleTriggerTransition		TransitionType
	ToTransition	SelectExecutionCycleToWaitForDiagnosticTriggerTransition		TransitionType
	ToTransition	SelectExecutionCycleToWaitForCleaningTriggerTransition		TransitionType
	ToTransition	SelectExecutionCycleToHoldingTransition		TransitionType
	ToTransition	SelectExecutionCycleToSuspendingTransition		TransitionType
WaitForCalibrationTrigger	HasProperty	StateNumber	200	PropertyType
	FromTransition	SelectExecutionCycleToWaitForCalibrationTriggerTransition		TransitionType
	ToTransition	WaitForCalibrationTriggerTo ExtractCalibrationSampleTransition		TransitionType
ExtractCalibrationSample	HasProperty	StateNumber	300	PropertyType
	FromTransition	WaitForCalibrationTriggerToExtractCalibrationSampleTransition		TransitionType
	ToTransition	ExtractCalibrationSampleToPrepareCalibrationSampleTransition		TransitionType
PrepareCalibrationSample	HasProperty	StateNumber	400	PropertyType
	FromTransition	ExtractCalibrationSampleToPrepareCalibrationSampleTransition		TransitionType
	ToTransition	PrepareCalibrationSampleToAnalyseCalibrationSampleTransition		TransitionType
AnalyseCalibrationSample	HasProperty	StateNumber	500	PropertyType
	FromTransition	PrepareCalibrationSampleToAnalyseCalibrationSampleTransition		TransitionType
	ToTransition	AnalyseCalibrationSampleToPublishResultsTransition		TransitionType
WaitForValidationTrigger	HasProperty	StateNumber	600	PropertyType
	FromTransition	SelectExecutionCycleToWaitForValidationTriggerTransition		TransitionType
	ToTransition	WaitForValidationTriggerToExtractValidationSampleTransition		TransitionType
ExtractValidationSample	HasProperty	StateNumber	700	PropertyType
	FromTransition	WaitForValidationTriggerToExtractValidationSampleTransition		TransitionType
	ToTransition	ExtractValidationSampleToPrepareValidationSampleTransition		TransitionType
PrepareValidationSample	HasProperty	StateNumber	800	PropertyType
	FromTransition	ExtractValidationSampleToPrepareValidationSampleTransition		TransitionType
	ToTransition	PrepareValidationSampleToAnalyseValidationSampleTransition		TransitionType

AnalyseValidationSample	HasProperty	StateNumber	900	PropertyType
	FromTransition	PrepareValidationSampleToAnalyseValidationSampleTransition		TransitionType
	ToTransition	AnalyseValidationSampleToPublishResultsTransition		TransitionType
WaitForSampleTrigger	HasProperty	StateNumber	1000	PropertyType
	FromTransition	SelectExecutionCycleToWaitForSampleTriggerTransition		TransitionType
	ToTransition	WaitForSampleTriggerToExtractSampleTransition		TransitionType
ExtractSample	HasProperty	StateNumber	1100	PropertyType
	FromTransition	WaitForSampleTriggerToExtractSampleTransition		TransitionType
	ToTransition	ExtractSampleToPrepareSampleTransition		TransitionType
PrepareSample	HasProperty	StateNumber	1200	PropertyType
	FromTransition	ExtractSampleToPrepareSampleTransition		TransitionType
	ToTransition	PrepareSampleToAnalyseSampleTransition		TransitionType
AnalyseSample	HasProperty	StateNumber	1300	PropertyType
	FromTransition	PrepareSampleToAnalyseSampleTransition		TransitionType
	ToTransition	AnalyseSampleToPublishResultsTransition		TransitionType
WaitForDiagnosticTrigger	HasProperty	StateNumber	1400	PropertyType
	FromTransition	SelectExecutionCycleToWaitForDiagnosticTriggerTransition		TransitionType
	ToTransition	WaitForDiagnosticTriggerToDiagnosticTransition		TransitionType
Diagnostic	HasProperty	StateNumber	1500	PropertyType
	FromTransition	WaitForDiagnosticTriggerToDiagnosticTransition		TransitionType
	ToTransition	DiagnosticToPublishResultsTransition		TransitionType
WaitForCleaningTrigger	HasProperty	StateNumber	1600	PropertyType
	FromTransition	SelectExecutionCycleToWaitForCleaningTriggerTransition		TransitionType
	ToTransition	WaitForCleaningTriggerToCleaningTransition		TransitionType
Cleaning	HasProperty	StateNumber	1700	PropertyType
	FromTransition	WaitForCleaningTriggerToCleaningTransition		TransitionType
	ToTransition	CleaningToPublishResultsTransition		TransitionType
PublishResults	HasProperty	StateNumber	1800	PropertyType
	FromTransition	AnalyseCalibrationToPublishResultsTransition		TransitionType
	FromTransition	AnalyseValidationToPublishResultsTransition		TransitionType
	FromTransition	AnalyseSampleToPublishResultsTransition		TransitionType
	FromTransition	DiagnosticToPublishResultsTransition		TransitionType
	FromTransition	CleaningToPublishResultsTransition		TransitionType
	ToTransition	PublishResultsToCleanupSamplingSystemTransition		TransitionType
	ToTransition	PublishResultsToEjectGrabSampleSystemTransition		TransitionType
EjectGrabSample	HasProperty	StateNumber	1900	PropertyType
	FromTransition	PublishResultsToEjectGrabSampleTransition		TransitionType
	ToTransition	EjectGrabSampleToCleanupSamplingSystemTransition		TransitionType

CleanupSamplingSystem	HasProperty	StateNumber	2000	PropertyType
	FromTransition	PublishResultsToCleanupSamplingSystemTransition		TransitionType
	FromTransition	EjectGrabSampleToCleanupSamplingSystemTransition		TransitionType
	ToTransition	CleanupSamplingSystemToSelectExecutionCycleTransition		TransitionType

5.3.4.5.4 *AnalyserChannel_OperatingModeExecuteSub*StateMachineType Transitions

Transitions are instances of *Objects* of the *TransitionType* defined in [UA Part 5] - SM Appendix which also includes the definitions of the ToState, FromState, HasCause, and HasEffect *Reference*s used. Table 79 specifies the Transitions defined for the *AnalyserChannel_OperatingModeExecuteSubStateMachineType.* Each Transition is assigned a unique *TransitionNumber*.

Table 79 – *AnalyserChannel_OperatingModeExecuteSub*StateMachine Transitions

BrowseName	**References**	**Target BrowseName**	**Value**	**Target Type Definition**	**Notes**
Transitions					
SelectExecutionCycleToWaitForCalibrationTriggerTransition	HasProperty	TransitionNumber	1	PropertyType	
	FromState	SelectExecutionCycle		StateType	
	ToState	WaitForCalibrationTrigger		StateType	
WaitForCalibrationTriggerToExtractCalibrationSampleTransition	HasProperty	TransitionNumber	2	PropertyType	
	FromState	WaitForCalibrationTrigger		StateType	
	ToState	ExtractCalibrationSample		StateType	
	HasCause	Trigger received			External cause
ExtractCalibrationSampleTransition	HasProperty	TransitionNumber	3	PropertyType	
	FromState	ExtractCalibrationSample		StateType	
	ToState	ExtractCalibrationSample		StateType	
ExtractCalibrationSampleToPrepareCalibrationSampleTransition	HasProperty	TransitionNumber	4	PropertyType	
	FromState	ExtractCalibrationSample		StateType	
	ToState	PrepareCalibrationSample		StateType	
PrepareCalibrationSampleTransition	HasProperty	TransitionNumber	5	PropertyType	
	FromState	PrepareCalibrationSample		StateType	
	ToState	PrepareCalibrationSample		StateType	
PrepareCalibrationSampleToAnalyseCalibrationSampleTransition	HasProperty	TransitionNumber	6	PropertyType	
	FromState	PrepareCalibrationSample		StateType	
	ToState	AnalyseCalibrationSample		StateType	
AnalyseCalibrationSampleTransition	HasProperty	TransitionNumber	7	PropertyType	
	FromState	AnalyseCalibrationSample		StateType	
	ToState	AnalyseCalibrationSample		StateType	
AnalyseCalibrationSampleToPublishResultsTransition	HasProperty	TransitionNumber	8	PropertyType	
	FromState	AnalyseCalibrationSample		StateType	
	ToState	PublishResults		StateType	
SelectExecutionCycleToWaitForValidationTriggerTransition	HasProperty	TransitionNumber	9	PropertyType	
	FromState	SelectExecutionCycle		StateType	
	ToState	WaitForValidationTrigger		StateType	
WaitForValidationTriggerToExtractValidationSampleTransition	HasProperty	TransitionNumber	10	PropertyType	
	FromState	WaitForValidationTrigger		StateType	
	ToState	ExtractValidationSample		StateType	
	HasCause	Trigger received			External cause
ExtractValidationSampleTransition	HasProperty	TransitionNumber	11	PropertyType	
	FromState	ExtractValidationSample		StateType	
	ToState	ExtractValidationSample		StateType	
ExtractValidationSampleToPrepareValidationSampleTransition	HasProperty	TransitionNumber	12	PropertyType	
	FromState	ExtractValidationSample		StateType	
	ToState	PrepareValidationSample		StateType	

BrowseName	References	Target BrowseName	Value	Target Type Definition	Notes
PrepareValidationSampleTransition	HasProperty	TransitionNumber	13	PropertyType	
	FromState	PrepareValidationSample		StateType	
	ToState	PrepareValidationSample		StateType	
PrepareValidationSampleToAnalyseValidationSampleTransition	HasProperty	TransitionNumber	14	PropertyType	
	FromState	PrepareValidationSample		StateType	
	ToState	AnalyseValidationSample		StateType	
AnalyseValidationSampleTransition	HasProperty	TransitionNumber	15	PropertyType	
	FromState	AnalyseValidationSample		StateType	
	ToState	AnalyseValidationSample		StateType	
AnalyseValidationSampleToPublishResultsTransition	HasProperty	TransitionNumber	16	PropertyType	
	FromState	AnalyseValidationSample		StateType	
	ToState	PublishResults		StateType	
SelectExecutionCycleToWaitFoSampleTriggerTransition	HasProperty	TransitionNumber	17	PropertyType	
	FromState	SelectExecutionCycle		StateType	
	ToState	WaitFoSampleTrigger		StateType	
WaitForSampleTriggerToExtractSampleTransition	HasProperty	TransitionNumber	18	PropertyType	
	FromState	WaitForSampleTrigger		StateType	
	ToState	ExtractSample		StateType	
	HasCause	Trigger received			External cause
ExtractSampleTransition	HasProperty	TransitionNumber	19	PropertyType	
	FromState	ExtractSample		StateType	
	ToState	ExtractSample		StateType	
ExtractSampleToPrepareSampleTransition	HasProperty	TransitionNumber	20	PropertyType	
	FromState	ExtractSample		StateType	
	ToState	PrepareSample		StateType	
PrepareSampleTransition	HasProperty	TransitionNumber	21	PropertyType	
	FromState	PrepareSample		StateType	
	ToState	PrepareSample		StateType	
PrepareSampleToAnalyseSampleTransition	HasProperty	TransitionNumber	22	PropertyType	
	FromState	PrepareSample		StateType	
	ToState	AnalyseSample		StateType	
AnalyseSampleTransition	HasProperty	TransitionNumber	23	PropertyType	
	FromState	AnalyseSample		StateType	
	ToState	AnalyseSample		StateType	
AnalyseSampleToPublishResultsTransition	HasProperty	TransitionNumber	24	PropertyType	
	FromState	AnalyseSample		StateType	
	ToState	PublishResults		StateType	
SelectExecutionCycleToWaitForDiagnostic TriggerTransition	HasProperty	TransitionNumber	25	PropertyType	
	FromState	SelectExecutionCycle		StateType	

BrowseName	References	Target BrowseName	Value	Target Type Definition	Notes
	ToState	WaitForDiagnostic Trigger		StateType	
WaitForDiagnosticTriggerToDiagnosticTr ansition	HasProperty	TransitionNumber	26	PropertyType	
	FromState	WaitForDiagnosticTrigger		StateType	
	ToState	Diagnostic		StateType	
	HasCause	Trigger received			External cause
DiagnosticTransition	HasProperty	TransitionNumber	27	PropertyType	
	FromState	Diagnostic		StateType	
	ToState	Diagnostic		StateType	
DiagnosticToPublishResultsTransition	HasProperty	TransitionNumber	28	PropertyType	
	FromState	Diagnostic		StateType	
	ToState	PublishResults		StateType	
SelectExecutionCycleToWaitForCleaning TriggerTransition	HasProperty	TransitionNumber	29	PropertyType	
	FromState	SelectExecutionCycle		StateType	
	ToState	WaitForCleaningTrigger		StateType	
WaitForCleaningTriggerToCleaningTrans ition	HasProperty	TransitionNumber	30	PropertyType	
	FromState	WaitForCleaningTrigger		StateType	
	ToState	Cleaning		StateType	
	HasCause	Trigger received			External cause
CleaningTransition	HasProperty	TransitionNumber	31	PropertyType	
	FromState	Cleaning		StateType	
	ToState	Cleaning		StateType	
CleaningToPublishResultsTransition	HasProperty	TransitionNumber	32	PropertyType	
	FromState	Cleaning		StateType	
	ToState	PublishResults		StateType	
PublishResultsToCleanupSamplingSyste mTransition	HasProperty	TransitionNumber	33	PropertyType	
	FromState	PublishResults		StateType	
	ToState	CleanupSamplingSystem		StateType	
PublishResultsToEjectGrabSampleTransi tion	HasProperty	TransitionNumber	34	PropertyType	
	FromState	PublishResults		StateType	
	ToState	EjectGrabSample		StateType	
EjectGrabSampleTransition	HasProperty	TransitionNumber	35	PropertyType	
	FromState	EjectGrabSample		StateType	
	ToState	EjectGrabSample		StateType	
EjectGrabSampleToCleanupSamplingSy stemTransition	HasProperty	TransitionNumber	36	PropertyType	
	FromState	EjectGrabSample		StateType	
	ToState	CleanupSamplingSystem		StateType	

BrowseName	References	Target BrowseName	Value	Target Type Definition	Notes
CleanupSamplingSystemTransition	HasProperty	TransitionNumber	37	PropertyType	
	FromState	CleanupSamplingSystem		StateType	
	ToState	CleanupSamplingSystem		StateType	
CleanupSamplingSystemToSelectExecutionCycleTransition	HasProperty	TransitionNumber	38	PropertyType	
	FromState	CleanupSamplingSystem		StateType	
	ToState	SelectExecutionCycle		StateType	
	HasCause	Configured acquisition is not completed			External cause

5.3.4.5.5 AnalyserChannel_OperatingModeExecuteSubStateMachineType Methods

There are no *Method*s defined for *AnalyserChannel_OperatingModeExecuteSubStateMachineType*.

5.3.4.6 AnalyserChannel_LocalModeSubStateMachineType

This specification does not define any sub-states for the AnalyserChannel_LocalModeSubStateMachineType.

5.3.4.7 AnalyserChannel_MaintenanceModeSubStateMachineType

This specification does not define any sub-states for the AnalyserChannel_MaintenanceModeSubStateMachineType.

5.3.5 AccessorySlotStateMachine

5.3.5.1 Introduction

The *AccessorySlotStateMachine* describes the behaviour of an *AccessorySlot* when a physical accessory is inserted or removed.

Figure 21 illustrates components of the *AccessorySlotStateMachineType.*

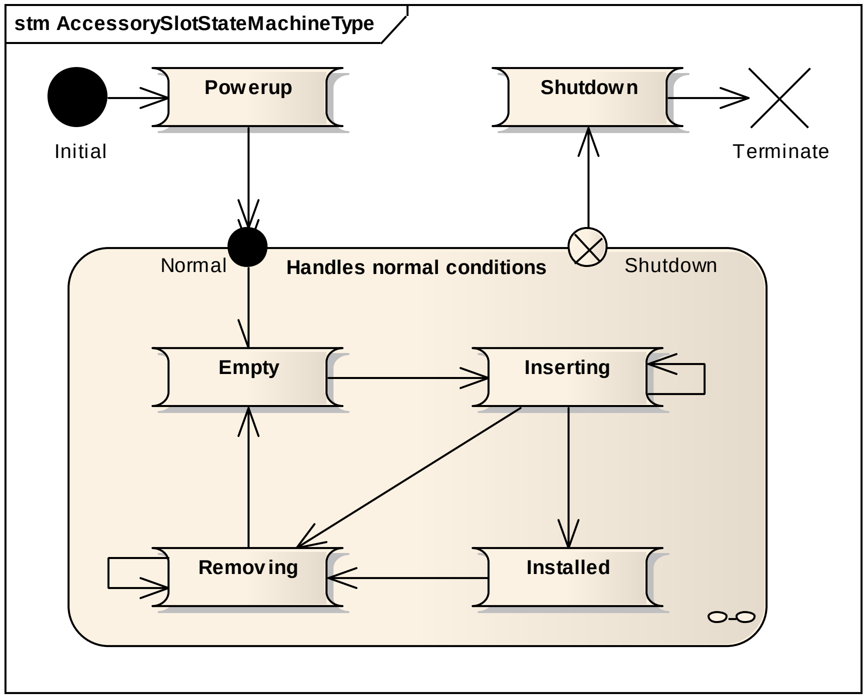

Figure 21 – AccessorySlotStateMachineTypeMachineType

If the accessory is not hot swappable or the accessory is already installed when the *AnalyserDevice* is powered-on the Inserting state becomes transient but remains present.

5.3.5.2 Type definition: AccessorySlotStateMachineType *ObjectType*

AccessorySlotStateMachineType is formally defined in Table 80.

Table 80 – AccessorySlotStateMachineType Definition

Attribute	Value				
	Includes all *Attributes* specified for the *FiniteStateMachineType*				
BrowseName	AccessorySlotStateMachineType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**Data Type**	**TypeDefinition**	**Modelling Rule**
Subtype of the *FiniteStateMachineType* defined in [UA Part 5]					
HasComponent	Object	Powerup		InitialStateType	Mandatory
HasComponent	Object	Empty		StateType	Mandatory
HasComponent	Object	Inserting		StateType	Mandatory
HasComponent	Object	Installed		StateType	Mandatory
HasComponent	Object	Removing		StateType	Mandatory
HasComponent	Object	Shutdown		StateType	Mandatory
HasComponent	Object	PowerupToEmptyTransition		TransitionType	Mandatory
HasComponent	Object	EmptyToInsertingTransition		TransitionType	Mandatory
HasComponent	Object	InsertingTransition		TransitionType	Mandatory
HasComponent	Object	InsertingToRemovingTransition		TransitionType	Mandatory
HasComponent	Object	InsertingToInstalledTransition		TransitionType	Mandatory
HasComponent	Object	InsttalledToRemovingTransition		TransitionType	Mandatory
HasComponent	Object	RemovingTransition		TransitionType	Mandatory
HasComponent	Object	RemovingToEmptyTransition		TransitionType	Mandatory
HasComponent	Object	EmptyToShutdownTransition		TransitionType	Mandatory
HasComponent	Object	InsertingToShutdownTransition		TransitionType	Mandatory
HasComponent	Object	InstalledToShutdownTransition		TransitionType	Mandatory
HasComponent	Object	RemovingToShutdownTransition		TransitionType	Mandatory

This specification does not define any *Methods*, which cause transitions in the *AccessorySlotStateMachineType*. Transitions occur as a result of two external causes:

- Accessory insertion
- Accessory removal

5.3.5.3 AccessorySlotStateMachineType States

Table 82 specifies the *AccessorySlotStateMachine's* State *Objects*. These State *Objects* are instances of the *StateType* defined in [UA Part 5] – Appendix B. Each State is assigned a unique *StateNumber* value. Subtypes of the *AccessorySlotStateMachineType* can add *References* from any state to a subordinate or nested *StateMachine Object* to extend the *FiniteStateMachine*.

A standard set of states are defined for *AccessorySlots*. These states represent the operational condition of the *AccessorySlot*. All *AccessorySlots* must support this base set. See Table 81 for the descriptions of the states.

Table 81 – AccessorySlotStateMachineType State Descriptions

StateName	Description
Powerup	The AccessorySlot is in its power-up sequence and cannot perform any other task.
Empty	This represents an AccessorySlot where no Accessory is installed.
Inserting	This represents an AccessorySlot when an Accessory is being inserted and initializing.
Installed	This represents an AccessorySlot where an Accessory is installed and ready to use
Empty	This represents an AccessorySlot where no Accessory is installed.
Shutdown	The AccessorySlot is in its power-down sequence and cannot perform any other task.

The set of states defined to describe an *AccessorySlot* can be expanded. Sub-states can be defined for the base states to provide more resolution to the process and to describe the cause and effects of additional stimuli and transitions. See Table 82 for the definitions of the states.

Table 82 – AccessorySlotStateMachineType States

BrowseName	**References**	**Target BrowseName**	**Value**	**Target TypeDefinition**	**Notes**
States					
Powerup	HasProperty	StateNumber	100	PropertyType	
	ToTransition	PowerupToEmptyTransition		TransitionType	
Empty	HasProperty	StateNumber	200	PropertyType	
	FromTransition	PowerupToEmptyTransition		TransitionType	
	FromTransition	RemovingToEmptyTransition		TransitionType	
	ToTransition	EmptyToInsertingTransition		TransitionType	
	ToTransition	EmptyToShutdownTransition		TransitionType	
Inserting	HasProperty	StateNumber	300	PropertyType	
	FromTransition	EmptyToInsertingTransition		TransitionType	
	ToTransition	InsertingToInstalledTransition		TransitionType	
	ToTransition	InsertingToRemovingTransition		TransitionType	
	ToTransition	InsertingToShutdownTransition		TransitionType	
Installed	HasProperty	StateNumber	400	PropertyType	
	FromTransition	InsertingToInstalledTransition		TransitionType	
	ToTransition	InstalledToRemovingTransition		TransitionType	
	ToTransition	InstalledToShutdownTransition		TransitionType	
Removing	HasProperty	StateNumber	500	PropertyType	
	FromTransition	InsertingToRemovingTransition		TransitionType	
	FromTransition	InstalledToRemovingTransition		TransitionType	
	ToTransition	RemovingToEmptyTransition		TransitionType	
	ToTransition	RemovingToShutdownTransition		TransitionType	
Shutdown	HasProperty	StateNumber	600	PropertyType	
	FromTransition	EmptyToShutdownTransition		TransitionType	
	FromTransition	InsertingToShutdownTransition		TransitionType	
	FromTransition	InstalledToShutdownTransition		TransitionType	
	FromTransition	RemovingToShutdownTransition		TransitionType	

Table 83 specifies the Transitions defined for the *AccessorySlotStateMachineType.* Each Transition is assigned a unique *TransitionNumber*.

Table 83 – AccessorySlotStateMachineType Transitions

BrowseName	References	Target BrowseName	Value	Target Type Definition	Notes
Transitions					
PowerupToEmptyTransition	HasProperty	TransitionNumber	1	PropertyType	
	FromState	Powerup		InitialStateType	
	ToState	Empty		StateType	
EmptyToInsertingTransition	HasProperty	TransitionNumber	2	PropertyType	
	FromState	Empty		StateType	
	ToState	Inserting		StateType	
InsertingTransition	HasProperty	TransitionNumber	3	PropertyType	
	FromState	Inserting		StateType	
	ToState	Inserting		StateType	
InsertingToRemovingTransition	HasProperty	TransitionNumber	4	PropertyType	
	FromState	Inserting		StateType	
	ToState	Removing		StateType	
InsertingToInstalledTransition	HasProperty	TransitionNumber	5	PropertyType	
	FromState	Inserting		StateType	
	ToState	Installed		StateType	
InstalledToRemovingTransition	HasProperty	TransitionNumber	6	PropertyType	
	FromState	Installed		StateType	
	ToState	Removing		StateType	
RemovingTransition	HasProperty	TransitionNumber	7	PropertyType	
	FromState	Removing		StateType	
	ToState	Removing		StateType	
RemovingToEmptyTransition	HasProperty	TransitionNumber	8	PropertyType	
	FromState	Removing		StateType	
	ToState	Empty		StateType	
EmptyToShutdownTransition	HasProperty	TransitionNumber	9	PropertyType	
	FromState	Empty		StateType	
	ToState	Shutdown		StateType	
InsertingToShutdownTransition	HasProperty	TransitionNumber	10	PropertyType	
	FromState	Inserting		StateType	
	ToState	Shutdown		StateType	
InstalledToShutdownTransition	HasProperty	TransitionNumber	11	PropertyType	
	FromState	Installed		StateType	
	ToState	Shutdown		StateType	
RemovingToShutdownTransition	HasProperty	TransitionNumber	12	PropertyType	
	FromState	Removing		StateType	
	ToState	Shutdown		StateType	

5.4 Variable Types

5.4.1 Introduction

OPC UA specification [UA Part 8] defines a *DataItem* as a link to arbitrary, live automation data, i.e. data that represents currently valid information. Examples of such data are: device data (such as temperature sensors), calculated data, status information (open/closed, moving), dynamically-changing system data (such as stock quotes), and diagnostic data.

AnalogItems are *DataItems* that represent continuously-variable physical quantities. Typical examples are the values provided by temperature sensors or pressure sensors. OPC UA defines *AnalogItemType VariableType* to identify an *AnalogItem*.

The ADI *Information Model* extends the *Variable* model defined in OPC UA specification [UA Part 3], [UA Part 5] and [UA Part 8], It introduces *VariableTypes*, which are specifically utilized for the process analytical domain.

5.4.2 Simple Types

Parameters which hold simple data like a single numerical value, string value or a time-stamp value are represented by *BaseDataVariableType* defined in [UA Part 5] or one of its subtypes.

For more details see paragraph C.1.

5.4.3 Array types

Parameters which hold array data that may be acquired during normal analyser operation or used as inputs (e.g. background, calibration) are represented by *VariableTypes,* which are direct subtypes of *DataItemType* and described in [UA Part 8].

For more details on *DataItemType* and its relationship with ADI *Parameters* see paragraph C.2

5.5 EngineeringValueType

The *EngineeringValue Variables* are used to expose key results of an analyser and the associated values that qualified it. This type helps the *Client* quickly identify important values. For example, the concentration of a given chemical and the associated confidence factors like the F-Ratio from the PLS model. *EngineeringValueType* is formally defined in Table 84

Table 84 – EngineeringValueType Definition

Attribute	**Value**				
BrowseName	EngineeringValueType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *DataItemType* defined in [UA Part 8]					
HasComponent	Variable	<Identifier>		DataItemType	OptionalPlaceHolder

The Value *Attribute* of the *EngineeringValue* is the main value, for example, the concentration. Its *HasComponent* elements are there to qualify or describe this value. For example the associated confidence factors like F-Ratio from the PLS model.

5.6 ChemometricModelType

The *ChemometricModel Variables* are used to hold the descriptions of a mathematical process and associated information to convert scaled data into one or more process values. ChemometricModelType is formally defined in Table 85.

All *ChemometricModel Variables* are located in the *ChemometricModelSettings FunctionalGroup* on a *Stream*.

Table 85 – ChemometricModelType Definition

Attribute	Value				
BrowseName	ChemometricModelType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *BaseDataVariableType* defined in [[UA Part 3]					
HasProperty	Variable	Name	LocalizedText	PropertyType	Mandatory
HasProperty	Variable	CreationDate	DateTime	PropertyType	Mandatory
HasProperty	Variable	ModelDescription	LocalizedText	PropertyType	Mandatory
HasInput	Variable	<User defined Input#>		BaseVariableType	MandatoryPlaceholder
HasOutput	Variable	<User defined Output#>		BaseVariableType	MandatoryPlaceHolder

Name is a descriptive name of the chemometric model itself e.g. *XYZ_Moisture_V1.0*.

CreationDate is the creation date of the chemometric model.

ModelDescription is a localized string describing the chemometric model itself e.g. *Predict the moisture in powder XYZ*.

HasInput is a subtype of *HasOrderedComponent Reference* which points to a *Variable*, defined in the *Analyser Server* address space, which is used as input for the chemometric model prediction. As a general rule, the target of *HasInput* is not instantiated at the *ChemometricModel* instantiation because it already exists elsewhere in the address space.

HasOutput is a subtype of *HasOrderedComponent Reference* which points to a *Variable* that is updated when the chemometric model is executed. As a general rule, the target of *HasOutput* is instantiated at the model instantiation because it is generated by the model itself. Often, the target of this *HasOuput Reference* is also the target of “Source” *Reference* of *ProcessVariable*.

Table 86 summarizes constraints on *Variable Attributes* and Properties for *ChemometricModelType*. For a complete set of *Attributes* see [UA Part 3], section 5.6.2.

Table 86 - Setting OPC UA Variable Attributes and Properties for ChemometricModelType

Attributes/Properties	Description
Value	Binary blob containing all elements of the chemometric model
DataType	ByteString
ValueRank	Always set to -1 (Scalar)
ArrayDimensions	Not applicable

5.7 ProcessVariableType

The *ProcessVariables* are used to provide a stable address space view from the user point of view even if the *Analyser Server* address space changes, after the new configuration is loaded. This is important to simplify integration with systems like DCS or LIMS that often require a stable mapping.

All *ProcessVariable Variables* are most of the time located in the *Stream AcquisitionData FunctionalGroup*. The location of the *ProcessVariable* can be found with these prioritized rules:

1) The location of a *ProcessVariable* shall remain constant between configurations. For example, if the number of *Streams* changes from one configuration to the other, the *ProcessVariables* shall be pushed one level up to the *AnalyserChannel*.

2) *ProcessVariable* should be located in the same *FunctionalGroup* as its *Source*.

The following bullets describe how the above rules should be applied to common scenarios:

- A typical lab analyser has one *AnalyserChannel* and one sample holder, which translates to a single *Stream*. In this case, *ProcessVariables* shall be located at the *Stream* level.

- A process analyser attached to a multi-port vessel with a fixed hardware setting, in this case also, *ProcessVariables* shall be located at the *Stream* level.

- A process analyser is installed on a dolly and can be attached to different vessels for diagnostic purposes. In this case, the number of *Streams* is likely to change from configuration to configuration. *ProcessVariables* shall be pushed to least *AnalyserChannel* level.

- An analyser publishes only a few values through *ProcessVariables* to mimic a legacy system. In this case, it may make sense to place ProcessVariables at the *AnalyserDevice* level.

- In gas chromatographs, new *Chromatographic Applications* (software *AnalyserChannels*) may be added over the time and similarly new *Streams* may be added or removed. Because these operations usually require hardware addition and they do not happen very often, it is strongly recommended to apply rule 2) to ensure the consistent way in which the control system views the gas chromatograph.

When a *ProcessVariable* is linked with another *Variable* through the *Source Reference*, it is the *Server*'s responsibility to copy and maintain in sync the following *Attributes* and *Properties* from the *Source* target:

- *Attributes*: *Value, DataType, ValueRank, ArrayDimensions, AccessLevel, UserAccessLevel, MinimumSamplingInterval*
- Standard *Properties*: *TimeZone, DayLightSavingTime, DictionaryFragment, AllowNulls* if they are present.

Knowing that the *ProcessVariables* are used to exchange values with control system, it is a good practice to keep the *DataType, ValueRank* and *ArrayDimensions* consistent between configurations.

Also, when the *Server* responds to read or *Subscription Services*, the returned *DataValue* shall be the same for both the *ProcessVariable* and the *Variable* pointed by the *Source Reference*, especially the *StatusCode, value and SourceTimestamp*.

ProcessVariableType is formally defined in Table 87.

Table 87 – ProcessVariableType Definition

Attribute	**Value**				
BrowseName	ProcessVariableType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *DataItemType* defined in [[UA Part 8]					
HasDataSource	Variable	<Source>		DataItemType (DataType defined by Source Variable)	Mandatory

Source is a *Reference* that usually points to an output *Variable* of a model but it is allowed to point to another *Variable*. The *DataType* of the *ProcessVariable* shall be the same as the one pointed by Source *Reference*.

5.8 Data Types

5.8.1 Introduction

The following paragraphs define the data types introduced by the ADI *Information Model*.

5.8.2 Enumerations

5.8.2.1 Introduction

Enumeration is used to represent a *Parameter* value that has a limited set of possible numeric values, each of which has a descriptive name. All *Parameters* of this kind are instances of *DataItemType VariableType*. The following definitions describe the values of the *EnumString Property* for those *Parameters* for the English locale (LocaleId=en).

5.8.2.2 ExecutionCycleEnumeration Type

ExecutionCycleEnumeration describes the type of acquisition cycle performed on a stream, in progress or completed.

Table 88 – ExecutionCycleEnumeration states

Seq. number	EnumString	Description
0	IDLE_0	No acquisition cycle in progress
1	DIAGNOSTIC_1	Diagnostic cycle
2	CLEANING_2	Cleaning cycle
3	CALIBRATION_4	Calibration cycle
4	VALIDATION_8	Validation cycle
5	SAMPLING_16	Normal Sample acquisition cycle
6	DIAGNOSTIC_WITH_GRAB_SAMPLE_32769	Diagnostic cycle with grab sample operation
7	CLEANING_WITH_GRAB_SAMPLE_32770	Cleaning cycle with grab sample operation
8	CALIBRATION_WITH_GRAB_SAMPLE_32772	Calibration cycle with grab sample operation
9	VALIDATION_WITH_GRAB_SAMPLE_32776	Validation cycle with grab sample operation
10	SAMPLING_WITH_GRAB_SAMPLE_32784	Normal Sample acquisition cycle with grab sample operation

When an ExecutionCycle with sequence number 6 through 10 (GRAB_SAMPLE) is selected, the operator or a system can grab a sample and send it to a control lab for analysis.

5.8.2.3 AcquisitionResultStatusEnumeration Type

AcquisitionResultStatusEnumeration describes acquisition result status on the *Stream* (general quality of the acquired data).

Table 89 – AcquisitionResultStatusEnumeration states

Seq. number	EnumString	Description
0	NOT_USED_0	No longer used.
1	GOOD_1	The acquisition has been completed as requested without any error.
2	BAD_2	The acquisition has been completed as requested with error.
3	UNKNOWN_3	The acquisition has been completed but nothing can be said about the quality of the result.
4	PARTIAL_4	The acquisition has been partially completed as requested without any error. For example, an averaging of 30 spectra as been requested, but the user terminates the acquisition after averaging 20 spectra.

5.9 Reference Types

5.9.1 HasDataSource

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

The semantic is a part-of relationship. The *TargetNode* of a *Reference* of the *HasDataSource ReferenceType* is providing the value for the *SourceNode*

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 *HasDataSource ReferenceType* is deleted when the *SourceNode* is deleted.

The source of the HasDataSource *ReferenceType* shall be of type *ProcessVariableType*.

There are no additional constraints defined for this *ReferenceType*.

5.9.2 HasInput

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

The semantic is a part-of relationship. The *TargetNode* of a *Reference* of the *HasInput ReferenceType* is providing an input value for a ChemometricModelType instance.

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 *HasInput ReferenceType* is deleted when the *SourceNode* is deleted.

The source of the HasInput *ReferenceType* shall be of type *ChemometricModelType*.

There are no additional constraints defined for this *ReferenceType*.

5.9.3 HasOutput

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

The semantic is a part-of relationship. The *TargetNode* of a *Reference* of the *HasOutput ReferenceType* is exposing an output value of a *ChemometricModelType* instance.

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 *HasOutput ReferenceType* is deleted when the *SourceNode* is deleted.

The source of the *HasOutput ReferenceType* shall be of type *ChemometricModelType*.

As a general rule, the target of *HasOutput ReferenceType* is a *DataVariable* generated by the *ChemometricModel* source.

There are no additional constraints defined for this *ReferenceType*.

6 Integration Profiles

This specification defines two OPC UA *Profiles* for an *Analyser Server* and a single OPC UA *Profile* for an *Analyser Client*. They are described in the following paragraphs.

6.1 Analyser Server Profiles

This specification defines two OPC UA *Profiles* for an Analyser Server. The *Profiles* are called *Level1 Analyser Server* and *Level2 Analyser Server*.

6.1.1 Level1 Analyser Server Profile

Level1 Analyser Server *Profile* includes the following standard *Profiles*, which are defined in [UA Part 7] and [UA-DI]. Those standard *Profile*s are mandatory components of Level1 Analyser Server *Profile*:

1) Embedded UA Server Profile
2) Auditing Server Facet
3) Basic Event Subscription Server Facet
4) ComplexType Server Facet
5) DataAccess Server Facet
6) Enhanced DataChange Subscription Server Facet.
7) Method Server Facet
8) UA-TCP UA-SC Binary Facet
9) BaseDevice_Server Facet.
10) DeviceIdentification_Server Facet

Table 90 describes *Conformance Units* included in Level1 Analyser Server *Profile*. These *Conformance Units* are in addition to the ones defined for standard *Profiles* listed above and defined in [UA Part 7].

Table 90 - Level1 Analyser Server Profile Conformance Units

Name	**Description**	**Optional/ Mandatory**
ADI Structures	Organization of the address space conforms to ADI specification. All mandatory components are included and referenced correctly.	M
ADI Parameters	All mandatory *Parameters* are present and located in the appropriate place in the ADI address space.	M
ADI Parameter Types	All *Parameters* have correct types.	M
ADI State Transitions	Only valid transitions and causes are allowed.	M
ADI Transition Events	All transitions generate events	M
ADI Methods	All *Method*s operate according to their descriptions and return valid results.	M
ADI Basic Configuration	SetConfiguration() and GetConfiguration() *Method*s successfully transfer complete Analyser Server configuration.	M

NOTE: All *Conformance Units* of Level1 Analyser Server *Profile* are self-testable. The complete list of published *Parameters* including those that can be used to configure *Analyser Server* shall be generated as part of the test certificate.

6.1.2 Level2 Analyser Server Profile

Level2 Analyser Server *Profile* includes Level1 Analyser Server *Profile*.

Table 91 describes *Conformance Units* included in Level2 Analyser Server *Profile*. These *Conformance Units* are in addition to the ones defined for Level1 Analyser Server *Profile*.

Table 91 - Level2 Analyser Server Profile Conformance Units

Name	Description	Optional/ Mandatory
ADI Advanced Configuration	Analyser Server exposes a complete set of read/write *Parameters* which can be used to configure the *Server*. The *Parameters* can be verified by comparing them with the vendor's proprietary configuration software or with that software's documentation.	M

6.2 Analyser Client Profile

This *Profile* includes the following standard *Profiles*, which are defined in [UA Part 7]. Those standard *Profiles* are mandatory components of Analyser *Client Profile*:

1) Core Client Profile
2) UA-TCP UA-SC Binary Facet
3) BaseDevice_Client Facet
4) DeviceIdentification_Client Facet

Table 92 describes additional *Conformance Units* applicable to the Analyser Client *Profile*. These *Conformance Units* are in addition to the ones defined for standard *Profiles* listed above and defined in [UA Part 7].

Table 92 - Analyser Client Profile Conformance Units

Name	Description	Optional/ Mandatory
ADI Complex Data	Analyser *Client* can interpret complex *Parameter* types and data types correctly.	O
ADI State Machine Display	Analyser *Client* can correctly visualize the ADI state machines.	O
ADI State Machine Control	Analyser *Client* can control the Analyser Server through its state machine.	O
ADI Configuration	Analyser *Client* can retrieve complete configuration from Analyser Server. Analyser *Client* can send and activate complete configuration to the Analyser Server	O

Annex A (informative) – Example of extending ADI Information Model for particle size monitor devices.

A.1 Overview

Analyser types which fall under the category of particle size monitor devices can extend the ADI model further by defining *Parameters* and/or subtypes of *ParticleSizeMonitorDeviceType*, *AccessoryType, AnalyserChannelType* or *StreamType*.

In the simplest case, no subtypes need to be defined and the *Parameters* can be exposed on existing *ParticleSizeMonitorDeviceType Object* or one of its components.

The following is an example of how a particle size monitoring device could extend the ADI *Information Model* by further refining definition of an *Accessory* and by defining new *Parameters* on *ParticleSizeMonitorType*.

A.2 Parameters of ParticleSizeMonitorDeviceType

A.2.1 AnalyserChannel of ParticleSizeMonitorDeviceType (Laser Diffraction Technology)

AnalyserChannelType defines two mandatory *FunctionalGroups* described in 5.2.2.1: *Configuration and Status. StreamType* defines seven mandatory *FunctionalGroups* described in 5.2.3.1: *Configuration, Status, AcquistionSettings, AcquisitionStatus, AcquisitionData, ChemometricModelSettings, and Context.* The following tables describe example sets of *Parameters* that can be defined on the *AnalyserChannel* and *Stream* of a *ParticleSizeMonitorDevice,* in this case using Laser Diffraction technology.

Table 93 – *ParticleSizeMonitorDeviceType AnalyserChannel* Configuration Parameters (Laser Diffraction Technology)

BrowseName	Description	VariableType	Optional/ Mandatory
DetectorCount	Number of detectors	DataItemType (DataType=Short)	O

Table 94 – *ParticleSizeMonitorDeviceType AnalyserChannel* Status Parameters (Laser Diffraction Technology)

BrowseName	Description	VariableType	Optional/ Mandatory
InstrumentConnected	Return the status of the physical connection with the channel	TwoStateDiscreteType (DataType=Boolean)	O
IsDetectorConnected	Return the status of the physical connection with the detector	TwoStateDiscreteType (DataType=Boolean)	O
IsLaserConnected	Return the status of the physical connection with the laser	TwoStateDiscreteType (DataType=Boolean)	O
IsLaserOn	Return the status of Laser (On/Off)	TwoStateDiscreteType (DataType=Boolean)	O

All *Parameters* organized by the *Status FunctionalGroup* on an *AnalyserChannel* of a *ParticleSizeMonitorDeviceType* shall be read-only.

Table 95 – *ParticleSizeMonitorDeviceStreamType* AcquisitionSettings Parameters (Laser Diffraction Technology)

BrowseName	Description	VariableType	Optional/ Mandatory
ParticleRI	Particle Refractive Index	DataItemType (DataType=RefractiveIndexType)	O
DispersantRI	Dispersant Refractive Index (Air, Water, Ethanol …)	DataItemType (DataType=RefractiveIndexType)	O
Density	Material density . (Kg/m3)	AnalogItemType (DataType=Double)	O
LowestDetector	Lowest detector enabled for acquisition	AnalogItemType (DataType=Short)	O
HighestDetector	Highest detector enabled for acquisition	AnalogItemType (DataType=Short)	O
Threshold	Minimum signal required for getting data	AnalogItemType (DataType=Double)	O
Gain	Detector gain	AnalogItemType (DataType=Double)	O
AnalysisType	Type of analysis. (Vendor Specific)	MultiStateDiscreteType (Vendor specific enumeration)	O
DistributionSizeLow	Minimum Size definition	AnalogItemType (DataType=Double)	O
DistributionSizeHigh	Maximum Size definition	AnalogItemType (DataType=Double)	O
DistributionChannelCount	Number of channel	AnalogItemType (DataType=Double)	O
ContinuousMode	Define the acquisition mode : Continuous Measurement Discontinuous Measurement	TwoStateDiscreteType (DataType=Boolean)	O
UpdatePeriod	In Continuous mode this is the period the analyser will produce a result	AnalogItemType (DataType=Float)	O
MeasurementDuration	In discontinuous mode this is the acquisition time	AnalogItemType (DataType=Float)	O
BackgroundDuration	Background measurement duration	AnalogItemType (DataType=Float)	O

A.2.2 AnalyserChannel of ParticleSizeMonitorDeviceType (General Approach)

As an alternative to chapter A.2.1, the tables below show a more general approach of defining the *Parameters*, independent of the used technology.

Table 96 – *ParticleSizeMonitorDeviceType AnalyserChannel* Status Parameters (Alternative to Table 94)

BrowseName	Description	VariableType	RW	Optional/ Mandatory
InstrumentConnected	Return the status of the physical connection with the channel	TwoStateDiscreteType (DataType=Boolean)	RO	O
ReadyForBackground	Return the status of the instrument and accessories to perform a background reading	TwoStateDiscreteType (DataType=Boolean)	RO	O
ReadyForMeasurement	Return the status of the instrument and accessories to perform a measurement	TwoStateDiscreteType (DataType=Boolean)	RO	O

All Parameters organized by the Status FunctionalGroup on an AnalyserChannel of a ParticleSizeMonitorDeviceType shall be read-only.

Table 97 – *ParticleSizeMonitorDeviceStreamType* AcquisitionSettings Parameters (Alternative to Table 95)

BrowseName	Description	VariableType	Optional/ Mandatory
AcquisitionSettings	Name of a set of acquisition settings stored in the analyser.	*String*	M

A.3 Accessories of ParticleSizeMonitorDeviceType

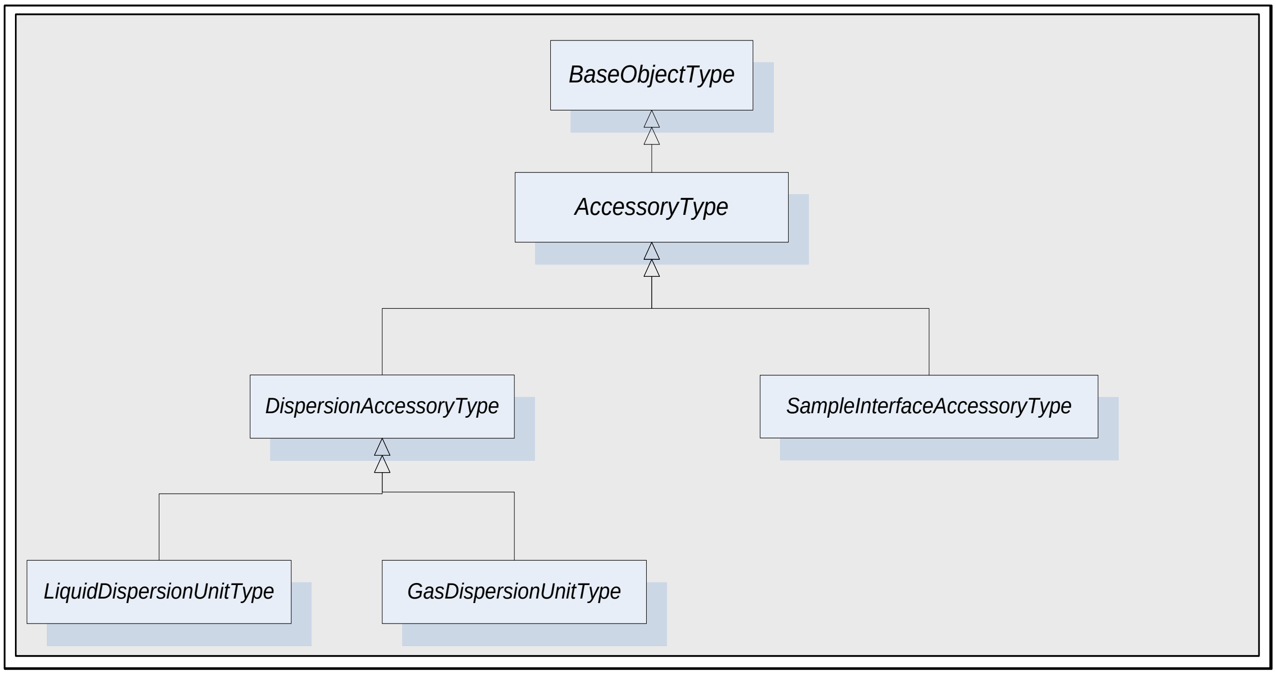

Figure 22 - AccessoryType of ParticleSizeMonitorDeviceType

DispersionAccessoryType is a subtype of *AccessoryType*. A dispersion unit allows dispersing powder. A dispersant is a liquid or gas added to a mixture to promote dispersion or to maintain dispersed particles in suspension.

LiquidDispersionUnitType is a subtype of *DispersionAccessory*. A liquid dispersion unit is a unit dispersing a mixture using a liquid (Water, ethanol …)

GasDispersionUnitType is a subtype of *DispersionAccessory*. A gas dispersion unit is a unit dispersing a mixture using a gas (Air, Nitrogen …)

SampleInterfaceAccessoryType is a subtype of *AccessoryType*. A sample interface is a unit allowing sample from the process line in order to perform a measurement. A sample interface accessory could be an auger, a rotational sampler, a simple probe, etc …

A.3.1 Type definition: DispersionAccessoryType *ObjectType*

Table 98 - DispersionAccessoryType

Attribute	Value				
BrowseName	DispersionAccessoryType				
IsAbstract	true				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AccessoryType* defined in 5.2.5.1					
HasSubtype	ObjectType	LiquidDispersionUnitType			
HasSubtype	ObjectType	GasDispersionUnitType			

A.3.2 Instance definition: DispersionAccessory *Object*

All *DispersionAccessoryType* have *Attributes* and *Properties* that they inherit from the *AccessoryType*. In addition to those, it is possible to define more *Parameters*.

A.3.2.1 Parameters of DispersionAccessoryType

DispersionAccessoryType can have, for example, the following *Parameters* defined.

Table 99 – DispersionAccessoryType Configuration Parameters

BrowseName	Description	VariableType	Optional/ Mandatory
DisperionSettings	Name of a set of dispersion settings stored in the analyser.	*string*	M

Table 100 – DispersionAccessoryType Status Parameters

BrowseName	Description	VariableType	Optional/ Mandatory
Mode	Accessory mode	MultiStateDiscreteType	O

All *Parameters* organized by the *Status FunctionalGroup* on a *DispersionAccessoryType* shall be read-only.

A.3.3 Subtypes of DispersionAccessoryType *ObjectType*

Subtypes of DispersionAccessoryType are optional. The definitions below serve as an example for Laser Diffraction or Image Analysers. Other technologies might require other definitions or none at all.

A.3.3.1 LiquidDispersionUnitType

A.3.3.1.1 Type definition: LiquidDispersionUnitType *ObjectType*

Table 101 - LiquidDispersionUnitType

Attribute	Value				
BrowseName	LiquidDispersionUnitType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *DispersionAccessoryType* defined in A.3.1.					

A.3.3.1.2 Instance definition: LiquidDispersionUnit *Object*

This *Object* defines an instance of the *LiquidDispersionUnitType* as defined in.

A.3.3.1.3 Parameters of LiquidDispersionUnitType

LiquidDispersionUnitType has the following *Parameters* defined.

Table 102 – LiquidDispersionUnitType Configuration Parameters

BrowseName	Description	VariableType	Optional/ Mandatory
PumpSpeed	Pump Speed allowing transporting the sample to the analyser	AnalogItemType (DataType=Double)	O
StirrerSpeed	Stirrer Speed allowing mixing sample and dispersant	AnalogItemType (DataType=Double)	O
Ultrasonic	Ultrasonic power allowing breaking agglomerate	AnalogItemType (DataType=Double)	O
UltrasonicMode	Apply ultrasonic continuously or periodically . (may be more option	MultiStateDiscreteType (Vendor specific enumeration)	O
UltrasonicTimeOn	Time the ultrasonic has to be ON	AnalogItemType (DataType=Double)	O
UltrasonicTimeOff	Time the ultrasonic has to be OFF	AnalogItemType (DataType=Double)	O

Table 103 – LiquidDispersionUnitType Status Parameters

BrowseName	Description	VariableType	Optional/ Mandatory
Mode	Accessory mode	MultiStateDiscreteType	O

All *Parameters* organized by the *Status FunctionalGroup* on a *LiquidDispersionUnitType* shall be read-only.

A.3.3.2 GasDispersionUnitType

A.3.3.2.1 Type definition: GasDispersionUnitType *ObjectType*

Table 104 – GasDispersionUnitType *Object*

Attribute	Value				
BrowseName	GasDispersionUnitType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *DispersionAccessoryType* defined in A.3.1					

A.3.3.2.2 Instance definition: GasDispersionUnit *Object*

This *Object* defines an instance of the *GasDispersionUnitType Object* as defined in Table 104

A.3.3.2.3 Parameters of GasDispersionUnitType

GasDispersionUnitType has the following *Parameters* defined.

Table 105 – GasDispersionUnitType Configuration Parameters

BrowseName	Description	VariableType	Optional/ Mandatory
Pressure	Pressure allowing dispersion	AnalogItemType (DataType=Double)	O
Flow	Gas flow for dispersing	AnalogItemType (DataType=Double)	O
FeedRate	Vibration Feeder	AnalogItemType (DataType=Double)	O

Table 106 - GasDispersionUnitType Status Parameters

BrowseName	Description	VariableType	Optional/ Mandatory
Mode	Accessory mode	MultiStateDiscreteType	O

All *Parameters* organized by the *Status FunctionalGroup* on a *GasDispersionUnitType* shall be read-only.

Annex B (informative) – Example of extending ADI Information Model for gas chromatograph devices

B.1 Overview

Analyser types which fall under the category of gas chromatographs (GC) can extend the ADI model further by defining *Parameters* and/or subtypes of *ChromatographDeviceType*, *Accessory*, *AnalyserChannel* and/or *Stream*.

In the simplest case, no subtypes of *ChromatographDeviceType* need to be defined and the *Parameters* can be exposed on existing *ChromatographDeviceType Object* or one of its components.

The following paragraphs provide an example of how a gas chromatograph device could extend the ADI *Information Model* by further refining definition of an *Accessory* and by defining new *Parameters* on *ChromatographDeviceType*.

The Figure 23 shows how a typical GC works:

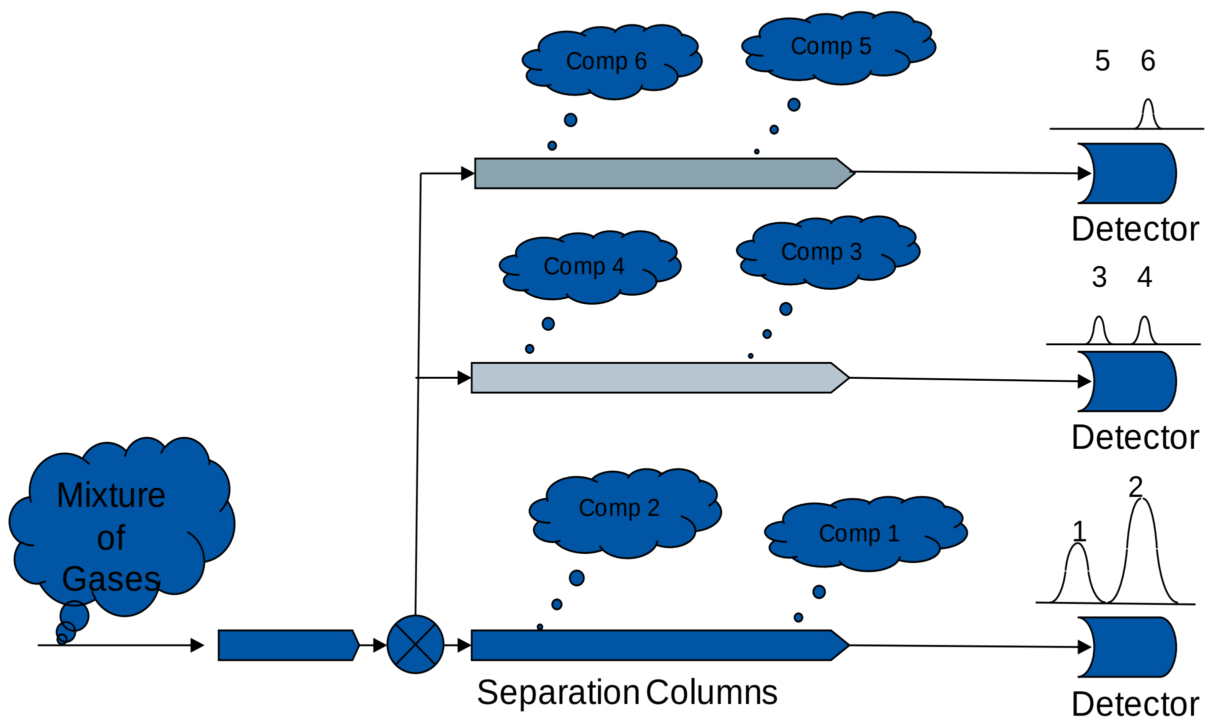

Figure 23 – GC overview

- The sample is extracted from the process using a sampling system external to the GC itself. It is done in a way that minimizes the time between the sample extraction from the process and its injection into the separation column sets.
- Each separation column set is maintained at a precise temperature by an oven, and used to separate molecules of the sample based on their size and chemical Properties.
- The propagation time through a given set of columns for a given component, is based on its size and Properties and the column Properties.

- The detector at the end of a column set monitors the outlet stream from the column; It determines the amount of a given component at the outlet and the time it used to reach it. The resulting detector output is a XY plot of the detector level versus time, called chromatogram.
- A set of mathematical algorithms converts the chromatogram peaks into component concentration.

B.2 Gas Chromatograph Parameters

B.2.1 Parameters defined for ChromatographDeviceType

The example set of *Parameters* defined on a *ChromatographDeviceType* for a gas chromatograph are described in Table 107 and Table 108.

Table 107- ChromatographDeviceType Configuration Parameters

BrowseName	Description	VariableType	Optional/ Mandatory
SetTime	SetTime is a Write only, Ole Date tag that is used to set the device and/or system time. If an analyser can be configured as a time server, the SetTime tag is used to update the time/date in that time server. The other devices in the system must be configured with the IP address of the designated time server.	DataItemType (DataType=DateTime)	O

Table 108 - ChromatographDeviceType Status Parameters

BrowseName	Description	VariableType	Optional/ Mandatory
ComState	The ComState tag is a read-only, 4-byte integer value that displays the current status of the communication link between the remote computer and the device.	DataItemType (DataType=Int32)	O

All *Parameters* organized by the *Status FunctionalGroup* on a *ChromatographDeviceType* shall be read-only.

B.2.2 Parameters defined for a *AnalyserChannel* of *ChromatographDeviceType*

The example set of *Parameters* defined on an *AnalyserChannel* of *ChromatographDeviceType* for a gas chromatograph are described in *Table 109*.

Table 109 **- ChromatographDeviceType AnalyserChannel Configuration Parameters**

BrowseName	Description	VariableType	Optional/ Mandatory
RunState	Sets the state of the chromatographic application. 0 = This application is in HOLD 1 = This application is in the RUN state 2 = This application is in the CALibration state 3 = This application is in the VALidation state Other values are not allowed. Write format: range 0 - 3 0 = Sets application to HOLD state 1 = Sets application to RUN state 2 = Sets application to CAL state 3 = Sets application to VAL state	*DataItemType (DataType=Int32)*	O

B.2.3 Parameters defined for a *ChromatographDeviceStreamType*

The example set of *Parameters* defined on a *ChromatographDeviceStreamType* for a gas chromatograph are described in Table 110.

Table 110 - ChromatographDeviceStreamType Configuration Parameters

BrowseName	Description	VariableType	Optional/ Mandatory
RunEvent	RunEvent is a read/write, 4-byte integer used to run a stream dependent event.	DataItemType (DataType=Int32)	O
SetAlarm	SetAlarm is a read/write, 4-byte integer that is used to set an alarm in the device.	DataItemType (DataType=Int32)	O

B.2.4 Representation of a gas chromatograph Component

The concentration of a given Component and its related characteristics are represented using a *Parameter* of a type derived from EngineeringValueType.

The *Value DataType* is *Float* and its *ValueRank* is -1 because it is a scalar.

All components of this *Variable* are read-only.

Table 111 and Table 112 illustrate two example definitions of types that represent gas chromatograph Components.

Table 111 - ABBComponentValueType definition

Attribute	Value				
BrowseName	ABBComponentValueType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**Description**	**TypeDefinition**	**ModellingRule**
Subtype of the EngineeringValueType					
HasComponent	Variable	AmplitudeEOB	Detector signal amplitude for the end of baseline calculation	AnalogItemType (DataType=Float)	Mandatory
HasComponent	Variable	AmplitudeEOI	Detector signal amplitude for the end of integration calculation	AnalogItemType (DataType=Float)	Mandatory
HasComponent	Variable	AmplitudeSOB	Detector signal amplitude for the start of baseline calculation	AnalogItemType (DataType=Float)	Mandatory
HasComponent	Variable	AmplitudeSOI	Detector signal amplitude for the start of integration calculation	AnalogItemType (DataType=Float)	Mandatory
HasComponent	Variable	BaselineEnd	Time into analysis of end of baseline calculation	AnalogItemType (DataType=Float)	Mandatory
HasComponent	Variable	BaselineStart	Time into analysis of start of baseline calculation	AnalogItemType (DataType=Float)	Mandatory
HasComponent	Variable	BenchmarkDeviation	This component's deviation from defined validation concentration	AnalogItemType (DataType=Float)	Mandatory
HasComponent	Variable	CrestAmplitude	Maximum peak height for this component	AnalogItemType (DataType=Float)	Mandatory
HasComponent	Variable	ExpectedConcentration	This component's expected concentration result from a validation analysis	AnalogItemType (DataType=Float)	Mandatory
HasComponent	Variable	ExpectedRT	This component's expected retention time for a given analysis	AnalogItemType (DataType=Float)	Mandatory
HasComponent	Variable	IntegrationEnd	Time into analysis of end of integration calculation	AnalogItemType (DataType=Float)	Mandatory
HasComponent	Variable	IntegrationStart	Time into analysis of start of integration calculation	AnalogItemType (DataType=Float)	Mandatory
HasComponent	Variable	IsValid	Validity flag for this component	TwoStateDiscreteType (DataType=Boolean)	Mandatory
HasComponent	Variable	NegativeArea	Negative peak area for this component	AnalogItemType (DataType=Float)	Mandatory
HasComponent	Variable	OldResponseFactor	Previous calibration response factor for this component	AnalogItemType (DataType=Float)	Mandatory
HasComponent	Variable	PeakFound	Flag for a peak detected	TwoStateDiscreteType (DataType=Boolean)	Mandatory
HasComponent	Variable	PositiveArea	Positive peak area for this component	AnalogItemType (DataType=Float)	Mandatory

HasComponent	Variable	ResponseFactor	Calibration response factor for this component and associated detector	AnalogItemType (DataType=Float)	Mandatory
HasComponent	Variable	Retention Time	Actual retention time at the peak apex for this component	AnalogItemType (DataType=Float)	Mandatory
HasComponent	Variable	RFUpdated	Flag if the new RF from a calibration analysis is accepted	TwoStateDiscreteType (DataType=Boolean)	Mandatory
HasComponent	Variable	TotalArea	Total peak area for this component	AnalogItemType (DataType=Float)	Mandatory

Table 112 – SiemensComponentValueType Definition

Attribute	Value				
BrowseName	SiemensComponentValueType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**Description**	**TypeDefinition**	**ModellingRule**
Subtype of the EngineeringValueType					
HasComponent	Variable	PkArea	Corrected peak area for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	PkArea%	Percent area for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	PkAsym	Peak asymmetry for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	PkAsym10	Asymmetry at 10% peak height for	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	PkHeight	Maximum peak height for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	PkHeight%	Percent height for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	PkNoise	Peak noise for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	PkResolution	Peak resolution relative to previous peak for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	PkRetTime	Retention time at peak apex for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	PkSignaltoNoise	Peak signal to noise for this component. (Peak height / rms noise)	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	PkStartFlg	Peak start flag for this component.	AnalogItemType (DataType=String)	Mandatory
HasComponent	Variable	PkStartTime	Retention time at start of peak for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	PkStopFlg	Peak stop flag for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	PkStopTime	Retention time at end of peak for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	PkType	Peak type this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	PkUSPWidth	Peak USP width (U.S. Pharmacopeia) for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	PkWidth	Peak width at base for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	PkWidth10	Peak width at 10% height for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	PkWidth5%	Peak width at 5% height for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	PkWidth50	Peak width at 50% height for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	PkTheorPlates	Theoretical plates for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	GrpArea	Corrected peak area (for the group) for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	GrpArea%	Peak area percent (for the group) for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	GrpHeight	Maximum peak height (for the group) for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	RespFactL0	Calibration level response factor for level 0 if applicable for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	RespFactL1	Calibration level response factor for level 1 if applicable for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	RespFactL2	Calibration level response factor for level 2 if applicable for this component.	AnalogItemType (DataType=Double)	Mandatory

HasComponent	Variable	RespFactL3	Calibration level response factor for level 3 if applicable for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	RespFactL4	Calibration level response factor for level 4 if applicable for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	RespFactL5	Calibration level response factor for level 5 if applicable for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	RespFactL6	Calibration level response factor for level 6 if applicable for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	RespFactL7	Calibration level response factor for level 7 if applicable for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	RespFactL8	Calibration level response factor for level 8 if applicable for this component.	AnalogItemType (DataType=Double)	Mandatory
HasComponent	Variable	RespFactL9	Calibration level response factor for level 9 if applicable for this component.	AnalogItemType (DataType=Double)	Mandatory

Annex C (informative) – Parameter Representation

C.1 Simple Parameters

Parameters which hold simple data like a single numerical value, string value or a time-stamp value are represented by *BaseDataVariableType* defined in [UA Part 5] or one of its subtypes.

*Variable*s which hold simple data like a single numerical value, string value or a time-stamp value are represented by *BaseDataVariableType* defined in [UA Part 5]. Those *Variable*s typically represent some configuration *Parameters*, status, states or acquisition results of an analyser.

If a *Variable* represents simple data which is obtained "live" from an analyser, *DataItem VariableType* or one of its subtypes will be used. For example, *AnalogItemType* shall be used when there is a need for a specific *Property* of the *AnalogItemType* such as *EURange* and *EngineeringUnits* [UA Part 8].

Table 72 describes how each *Attribute* of super/sub class of *DataItem* is used in the ADI context.

Table 113 - ADI DataItem Attributes

Attributes/Properties	Description
Base NodeClass	
DisplayName	Localized user readable name of this DataItem
BrowseName	The programmatic name of this DataItem
Description	Localized user readable description
WriteMask	Supports access control implementation
UserWriteMask	Supports access control implementation
Variable NodeClass	
Value	The *Parameter* value itself
DataType	DataType of Value
ValueRank	The number of dimensions of value
ArrayDimensions	The size of each value dimensions
AccessLevel	Supports access control implementation of Value
UserAccessLevel	Supports access control implementation of Value
MinimumSamplingInterval	Defined how fast Value may be updated
DataItemType	
Definition	Vendor-specific, human readable string that specifies how the value of this DataItem is calculated. Definition is non-localized and will often contain an equation that can be parsed by certain *Client*s. Example Definition ::= "(TempA – 25) + TempB"
ValuePrecision	The maximum precision that the *Server* can maintain for the item based on restrictions in the target environment
AnalogItemType	
InstrumentRange	The value range that can be returned by the instrument
EURange	The value range likely to be obtained in normal operation. It is intended for such use as automatically scaling a bar graph display
EngineeringUnits	The units for the DataItem's value (e.g., DEGC, hertz, seconds)
TwoStateDiscreteItemType	
TrueState	String to be associated with this DataItem when it is TRUE. This is typically used for a contact when it is in the closed (non-zero) state. e.g. "RUN", "CLOSE", "ENABLE", "SAFE", etc.
FalseState	String to be associated with this DataItem when it is FALSE. This is typically used for a contact when it is in the open (zero) state. e.g. "STOP", "OPEN", "DISABLE", "UNSAFE", etc.
MultiStateDiscreteItemType	
EnumStrings	EnumStrings which is a string lookup table corresponding to sequential numeric values (0, 1, 2, etc.) Example: "OPEN" "CLOSE" "IN TRANSIT" etc

The other source of information is the OPC UA Read *Service* described in [UA Part 4]. It provides:

- The current value itself
- Quality of the value
- The time of the last update

The SemanticsChanged bit in *StatusCode*

C.2 Array Parameters

Parameters which hold array data that may be acquired during normal analyser operation or used as inputs (e.g. background, calibration) are represented by VariableTypes which are direct subtypes of *DataItemType*.

They inherit all of the Properties of the *DataItemType*. Also, they inherit a set o *Attributes* from the *Variable* NodeClass that are common to all derived VariableTypes. Refer to Table 113 for more information.

The decision to base the array types on the *DataItemType* rather than the *AnalogItemType* is to allow a modification of the *StatusCode*.SemanticsChanged. In the *AnalogItemType*, this bit is set if and only if a change occurs in one or several of the Properties *InstrumentRange*, *EURange* and *EngineeringUnits*. In the ADI array types; this bit changes if and only if a change occurs in one or several of the Properties *InstrumentRange*, *EURange*, *EngineeringUnits* and the axis definitions. This also allows the implementation of the type where the Value.DataType is not a subclass of Number like in the XYArrayItemType.

To simplify the development of ADI *Client*s/*Server*s, the Properties *InstrumentRange*, *EURange* and *EngineeringUnits*, that are part of the *AnalogItemType* definition, are reused with exactly the same semantic as in the *AnalogItemType*:

InstrumentRange defines the value range that can be returned by the instrument.

Example: InstrumentRange ::= {-9999.9, 9999.9}

The *Range* type is specified in [UA Part 8].

EURange defines the value range likely to be obtained in normal operation. It is intended for such use as automatically scaling a bar graph display.
Sensor or instrument failure or deactivation can result in a returned item value which is actually outside this range. *Client* software must be prepared to deal with this. Similarly a *Client* may attempt to write a value that is outside this range back to the *Server*. The exact behaviour (accept, reject, clamp, etc.) in this case is *Server*-dependent. However in general *Server*s must be prepared to handle this.

Example: EURange ::= {-200.0,1400.0}

EngineeringUnits specifies the units for the *DataItem*’s value (e.g., DEGC, hertz, seconds).

The *EUInformation* type is specified in [UA Part 8].

If the item contains an array the Properties will apply to all elements in the array.

Annex D (informative) – Events, Alarms and Conditions

This specification does not introduce any standard types of *Events*, *Alarms* or *Conditions*.

Transitions defined as part of the *AnalyserDeviceStateMachineType*, *OperatingStateMachineType*, their subtypes and sub-states shall produce events which are subtypes of *TransitionEventType* defined in [UA Part 5].

Annex E (informative) – Operation level result codes

Table 114 provides additional ADI-specific guidelines for interpretation of the *Uncertain* operation level result code defined in [UA Part 8].

Table 114 - Uncertain operation level result codes

Symbolic Id	**Description**
Uncertain_ NoCommunicationLastUsable	Communication to the data source has failed. The *Variable* value is the last value that had a good quality and it is uncertain whether this value is still current. The *Server* timestamp in this case is the last time that the communication status was checked. The time at which the value was last verified to be true is no longer available. In ADI, this implies that the communication to the analyser has failed, but the *Analyser Server* is still active and communicating with its *Client*s. The *Client*s need updates, so the *Server* is responsible for maintaining the namespace and all the values.
Uncertain_ LastUsableValue	Whatever was updating this value has stopped doing so. This happens when an input *Variable* is configured to receive its value from another *Variable* and this configuration is cleared after one or more values have been received. This status/substatus is not used to indicate that a value is stale. Stale data can be detected by the *Client* looking at the timestamps. In ADI, this differs from the **Uncertain_NoCommunicationLastUsable** code only in that is does not explicitly state that there is no communication. For some undetermined reason, the analyser can no longer update the values. In the case of spectrographic analysers, there may be a significant error in the model that stops the collection and analysis (too many bad scans, divide by zero exception in the math model, etc.).
Uncertain_SubstituteValue	The value is an operational value that was manually overwritten. This value is a placeholder value that is set by the user when the instrument cannot collect or update the data.
Uncertain_InitialValue	The value is an initial value for a *Variable* that normally receives its value from another *Variable*. This status/substatus is set only during configuration while the *Variable* is not operational (while it is out-of-service). In ADI, this bit is set for all *Variable*s when the configuration is first loaded and started. The initial value is a preconfigured value defined when the instrument is first configured.
Uncertain_ SensorNotAccurate	The value is at one of the sensor limits. The Limits bits define which limit has been reached. Also set if the device can determine that the sensor has reduced accuracy (e.g. degraded analyser), in which case the Limits bits indicate that the value is not limited. In ADI, some internal diagnostic value in the analyser indicates that there is something inaccurate or untrustworthy in the data. For example, in FTIR, the interferogram peak center burst location or height may be beyond the acceptable threshold. Also, the internal temperature of the analyser may be out of specification. In both cases, spectra can be collected, but the accuracy of those spectra are in doubt.
Uncertain_ EngineeringUnitsExceeded	The value is outside of the range of values defined for this *Parameter*. The Limits bits indicate which limit has been reached or exceeded. In ADI, there are multiple contexts where this code is applicable. In the instrument, it is possible that the analyser sensor or detector is close to saturated or overexposed. The analyser hardware itself is almost incapable of measuring the physical system, and thus any results from the analyser are untrustworthy. For example, if the detector saturates at 32767 counts, any readings over 28000 counts can be deemed uncertain. These limits are vendor specific. Another example involves the mathematical modelling that occurs in the analysers. Analysers are typically calibrated and optimized to measure data and produce results in a particular range. If the inputs or calculated output exceeds that range, the validity of the mathematical calculations and results are uncertain.
Uncertain_SubNormal	The value is derived from multiple sources and has less than the required number of `Good` sources. In the analyser, the data may be an accumulation or an averaging of many measurements. If any of those measurements is uncertain or bad, then the data is subnormal. For example, spectra are typically a co-addition or an averaging of multiple scans of the system. It is possible that some, but not all of those scans, may be bad or unusable. A few unusable scans will not prevent the system from collecting and processing the data. However, the fact that some

	bad scans exist should not be ignored either. If the number of bad scans exceeds a vendor defined threshold, then the data is subnormal.

Annex F (informative) – ADI address space

This annex is intended to provide some guidelines on how to design the address space of an *Analyser Server*. It covers the following topics:

- Main questions to answer before starting the address space design
- Configuration process
- Parameter definition
- OPC UA key elements
- General rules

This annex should be used as a check list of points to address during the design and the source of description of the rationale behind some elements of the ADI specification. The annex is written in the FAQ format.

F.1 Define your Analyser Server

This section describes the basic questions that must be answered before starting the address space design.

1) Is the number of *AnalyserChannels* constant for this analyser or does it change frequently? For example, the GC has the concept of software *AnalyserChannel*, and new *AnalyserChannels* may be added over the time.
2) Is the number of *Streams* per *AnalyserChannel* constant for this analyser or does it change frequently?
3) Does the analyzer have a default configuration that allows the user to start the analyser without loading a configuration? This is true for small dedicated analysers that have a single mode of operation. This can also be true if the last configuration is automatically recalled at analyser power-up.
4) Will the analyser continue to acquire data if the connection with the client is broken?
5) Does the analyser server implement access control?
   a) What is the access control scheme: user id based, role based?
6) Does the analyser server expose the same *Parameters* to all users?
7) Knowing that this specification has been developed partly to support analyser deployment in the pharmaceutical industry, how do you plan to support 21 Part 11 regulations? This is not mandatory in the ADI specification, but a good practice to plan for it knowing that it does not require more development, but rather follow some basic design concepts.
8) How do you plan to do the internationalization of the text elements that can be translated?
9) What is your error handling philosophy?
   a) How do you report error to the client: through status, events ...?
   b) What do you put in the audit log?
10) Do you support more than one model of analyzer with this analyser server?
11) Does this analyser server handle more than one analyze simultaneously?

F.2 Configuration

This section addresses questions related to the analyser server configuration.

1) What is the analyser server configuration philosophy?

a) Offline configuration using vendor specific tool using a proprietary configuration format, then call *SetConfiguration* method. This approach is often cost effective in the short term, when migrating legacy analysers, but limits the benefits of the open ADI standard.

b) Offline configuration using vendor specific tool but using a public, documented configuration format. A documented XML format is a good example.

c) Start the analyser server and configure each *Parameter* manually using an OPC UA / ADI client.

d) Dual approach, which combines the offline configuration with public configuration format followed by a call to SetConfiguration. This can be followed with the client updating some *Parameters* before starting the acquisition process. This is the preferred approach because it allows the client to:

   i) Use standard configuration templates and apply specific changes.

   ii) Use generic tools for configuration and deployment tasks

   iii) Update some *Parameters* live, which is very convenient during diagnostics.

F.3 Parameters

This section helps define, initialise and position *Parameters* in the *Analyser Server* address space.

F.3.1 What is a Parameter?

1) A configuration parameter defining one of the settings of the hardware of the analyser.
2) A configuration parameter defining one of the settings of the behaviour of the analyser.
3) A status parameter exposing the state / status of the hardware of the analyser i.e. power supply temperature.
4) A status parameter exposing the state / status of the behaviour of the analyser i.e. which *Stream* is active?
5) A configuration parameter defining one of the settings of the hardware of the analyser for the current acquisition or the one to be started i.e. gain of a detector.
6) A configuration parameter defining one of the settings of the behaviour of the analyser for the current acquisition or the one to be started i.e.: duration of the acquisition.
7) A status parameter exposing the state / status of the hardware of the analyser for the current acquisition or the one to be started i.e. detector gain too high.
8) A status parameter exposing the state / status of the behaviour of the analyser for the current acquisition or the one to be started i.e. % done of the ongoing acquisition.
9) A result parameter exposing results generated by the analyser or derived from it i.e. absorbance spectrum, particle size distribution, concentration.
10) A *ChemometricModel* parameter exposing the model definition used to convert ScaledData to derived properties e.g. concentration derived from the absorbance spectrum.
11) An input I/O from a PLC indicating when the sample is ready.
12) An output I/O telling the sampling system that it can now grab a sample from the process.

F.3.2 Which Parameters should be exposed?

This question is very important because even if you can expose a *Parameter*, this does not mean that you should do so. Providing too many *Parameters* will create a complex address space for no good reason. The following questions should be asked:

1) Who will use this Parameter?

   a) The end users like the acquisition results.
   b) The configuration tools like the acquisition configuration *Parameters.*
   c) The analyser vendor production people may like setting the serial number.
   d) The service personnel may like internal diagnostics.
   e) R&D people may like some obscure servo loop control Parameter
2) Which client system component will record the *Parameter*?
   a) The plant DCS likes the concentration
   b) The Historian likes absorbance spectrum and concentration
   c) The Asset management likes expected remaining life of IR source or laser.

F.3.3 Parameter type

This section answers some common questions regarding the types of Parameters.

1) All Parameters should be derived from DataItemType
2) Try to use types defined in the ADI specification, they have been defined specifically for analyser data.
3) Try to use types defined in DI specification, they have been defined to standardize device *Parameters*.
4) Try to use types defined in OPC UA specification
5) If none of the predefined types are appropriate, derive a new type from one of the existing ones. This approach allows generic clients to handle them more easily.
6) Use standard *Properties* when appropriate. This allows generic clients to handle them more easily.
7) Define *EngineeringUnits* where appropriate. This is very important from a user perspective to know what he/she is dealing with.
8) Set *Description* and *Definition Attributes* to allow *Analyser Server* browsing and to help generic clients understand what they are looking at.
9) Set *EURange* and *InstrumentRange* where appropriate to help generic clients better interpret the results.
10) For Boolean *Parameters*, consider using *TwoStateDiscreteType* to provide useful names for True and False values.

F.3.4 Parameter attributes and standard properties

This section aims to provide help in deciding what values should be assigned to *Parameter Attributes* and standard *Properties.*

1) BrowseName

   English name of the Parameter, which is used for programmatic purpose only. It is never shown to the user. You should avoid using “_” character because it may clash with some development tools.

2) AccessLevel

   Definethis *Attribute* if this *Parameter’s* value is Read-Only or Read/Write independent of the user access rights. In general, this value is constant except if it depends of the state of the analyser.

3) UserAccessLevel

   Define this *Attribute* if this *Parameter’s* value is ReadOnly or Read/Write based on the user access rights of the user who is trying to access it. The server shall update this attribute at runtime based on who is logged in.

4) WriteMask

Define this *Attribute* if this *Parameter's* attributes are ReadOnly or Read/Write independent of the user access rights. In general, this value is constant except if it depends of the state of the analyser. If the server can always provide a good value, there is no need to bother the user with it.

5) UserWriteMask

Define this *Attribute* if this *Parameter's* attributes are ReadOnly or Read/Write based on the user access rights of the user who is trying to access it. The server shall update this *Attribute* at runtime based on who is logged in. If the server can always provide a good value, there is no need to bother the user with it.

6) MinimalSamplingInterval

Define at which rate the server monitors / updates the value of this *Parameter*.

a) If the server never updates this *Parameter* by itself, there is no need to define MinimalSamplingInterval.

b) If the server updates this *Parameter*, MinimalSamplingInterval should be initialized with a value based on the rate at which the analyzer will update the *Parameter*.

c) Do you want to allow the user to set this value or let the server decide? If yes, WriteMask and UserWriteMask shall be set. In any case, a reasonable initial value shall be provided.

F.3.5 Parameter FunctionalGroup

This section aims to provide a set of guidelines for deciding in which *FunctionalGroup* a given *Parameter* should be located:

1) If it is common to all *AnalyserChannels*, it should be at the *AnalyserDevice* level.
2) If it is common to all *Streams* of a given *AnalyserChannel*, it should be at the *AnalyserChannel* level.
3) If it is different for each *Stream*, it shall be at the *Stream* level.
4) If it is a configuration *Parameter* that does not change from acquisition to acquisition, it should be in the *Configuration FunctionalGroup*.
5) If it is a *Parameter* that is not intended to be modified by the user, is should be in the *FactorySettings* e.g. SpectralRange of the analyzer, which it is defined at the factory.
6) If the *Parameter* changes for each acquisition, it should be in *AcquisitionSettings* e.g. DetectorGain.
7) If the *Parameter* describes the setting of the current acquisition or the one to be started, it should be in *AcquisitionSettings* e.g. NumberOfScansToBeDone.
8) If the *Parameter* is an input from an external system like a PLC, and used to control the acquisition cycle, it should be in *AcquisitionSettings* e.g. AcquisitionTrigger.
9) If a status *Parameter* is independent of the acquisition in progress, it should be in a Status *FunctionalGroup* e.g. DiagnosticStatus.
10) If a status *Parameter* changes during the acquisition, it should be in the *AcquisitionStatus FunctionalGroup* e.g. Progress.
11) If the *Parameter* is updated at the end of each acquisition, it should be in *AcquisitionData* e.g. ScaledData.
12) If the *Parameter* is derived from a *Parameter* in *AcquisitionData*, it should also be in

AcquisitionData e.g. the concentration derived from the absorbance spectrum.

ADI specification does not define when the *Parameters* in the *AcquisitionSettings FunctionalGroup* should be changed. As a general rule they should not change after the start of acquisition except for a case involving an acquisition trigger.

F.3.6 Validation rules

It is a good practice to define the validation rules for each *Parameter*. For example:

1) Valid range
2) List of possible values
3) Cross-*Parameter* validation rules e.g. MinFrequency shall be smaller than MaxFrequency
4) Cross-*FunctionalGroup* validation rules e.g. if the analyzer is in MidIR configuration, MaxFrequency shall be smaller the 7899cm^{-1}.
5) A consistent way of defining these validating rules is important for the ability to write generic ADI tools.

F.4 Methods

It is mandatory to correctly set (at runtime) the Executable and UserExecutable attributes to indicate if a *Method* may be called in the current state of the *Analyser Server* and if the currently logged-in user may request its execution.

Vendors have the right to add custom methods to each component of the system. For example:

1) LoadFirmware *Method* at the *AnalyserDevice* level.
2) MoveToNextSample *Method* on a multi-sample holder accessory.

All *Methods* shall be located on the *MethodSet Object* when the component is a *TopologyElement*.

F.5 DeviceType properties

The following *Properties* need to be initialized on startup of the analyser: SerialNumber, RevisionCounter, Model, Manufacturer, DeviceManual, DeviceRevision, SoftwareRevision and HardwareRevision. The following rules apply:

1) If the *Analyser Server* handles more than one model, their values need to be extracted from the analyser itself.
2) The RevisionCounter should be updated under following circumstances:
   a) After each call of LoadFirmware if it exists.
   b) When a hardware element is replaced.

F.6 Disconnection handling

Knowing that the connection between the *Analyser Server* and the client may be broken for a short period (e.g. WiFi signal drop), it is wise to plan for it. If the analyser will continue to acquire data when the connection with the client is broken:

1) Appropriate subscription length of the FIFO queue shall be allowed, based on the analyzer output rate and the maximum permitted disconnection time. This shall be set consistently across the whole *AcquisitionData FunctionalGroup*
2) In general, only *AcquisitionData* needs a subscription FIFO queue.
3) The client settings should be:

   a) Subscription FIFO KeepOldest flag should be set to true.
   b) Client should keep track of dropped packets and request Republish for the missing ones.

4) The server does not have to do any throttling because the client controls the flow through the Publish request.
5) The size of the re-publish queue should be evaluated based on the type of the disconnection.

Annex G (informative) – Prediction service

This annex is intended to provide some guidelines on how to expose a ChemometricModel prediction service on an existing ADI server.

G.1 Prediction server use case

G.1.1 Description

The ADI Prediction server is a standard ADI server offering a service allowing:

- A client application to load models that are visible to any or selected applications.
- Many applications to request concurrently different predictions from the same or multiple models.
- Many models to reside at the same time in the ADI Prediction server cache, speeding up the prediction evaluation.
- The pre-loaded models definitions to be discovered by navigating the Address Space of the ADI Prediction server.

G.1.2 Main success scenario

Scenario steps	Interface used
1. The client application logs on ADI Prediction server	OPC UA CreateSession
2. The client application reads model from disk and loads them in ADI Prediction server	OPC UA AddNode and write services
3. ADI Prediction server exposes all model structures in its address space	OPC UA address space + ADI data types
4. The client application logs off ADI Predictor server	OPC UA CloseSession
5. The same or another client application logs on ADI Prediction server	OPC UA CreateSession
6. Client application identifies loaded models and their structure	OPC UA Browse address space + ADI data types
7. Client application calls MVAPredict () method with the appropriate input parameters,	OPC UA/ADI method call service + ADI data types
8. ADI Prediction server computes predicted values using vendor native libraries.	Vendor specific native libraries
9. Application reads predicted values from MVAPredict () method output parameters.	OPC UA/ADI method call service + ADI data types
10. Repeat steps 7 to 9 for each model	
11. Application logs off from ADI Prediction server	OPC UA CloseSession

Many applications may execute Steps 5 to 11 concurrently.

G.1.2.1 Extensions

1. The client application may pass the input values and received predicted values using OPC UA/ADI address space. However, this approach does not scale up well in multi-user environment because a prediction server is expected to serve many users concurrently which make synchronization almost impossible.
2. In custom / dedicated configurations, the ADI Prediction server loads a predefined set of models, so client applications can use them immediately.

G.2 Prediction service

To expose a prediction service on an existing ADI server independent of the rest of the ADI server:

1. Create a MVAPredictMethodType instance named MVA*Predict* on the AnalyserDevice or its derived object on the ADI server.

2. Add a HasComponent reference from the AnalyserDevice.MethodSet object to the MVA*Predict method*.

3. Create a FunctionalGroup named *ChemometricModelSettings* on the AnalyserDevice or its derived object of the ADI server.

4. Add a HasComponent reference from the AnalyserDevice object to the *ChemometricModelSettings* FunctionalGroup.

5. Create MVAModelType objects in the *ChemometricModelSettings* FunctionalGroup.

6. Add a HasComponent references from the *ChemometricModelSettings* to each MVAModelType object.

From that point, models may be predicted individually using MVAPredict method. Models may also be updated using the OPC UA Write service.

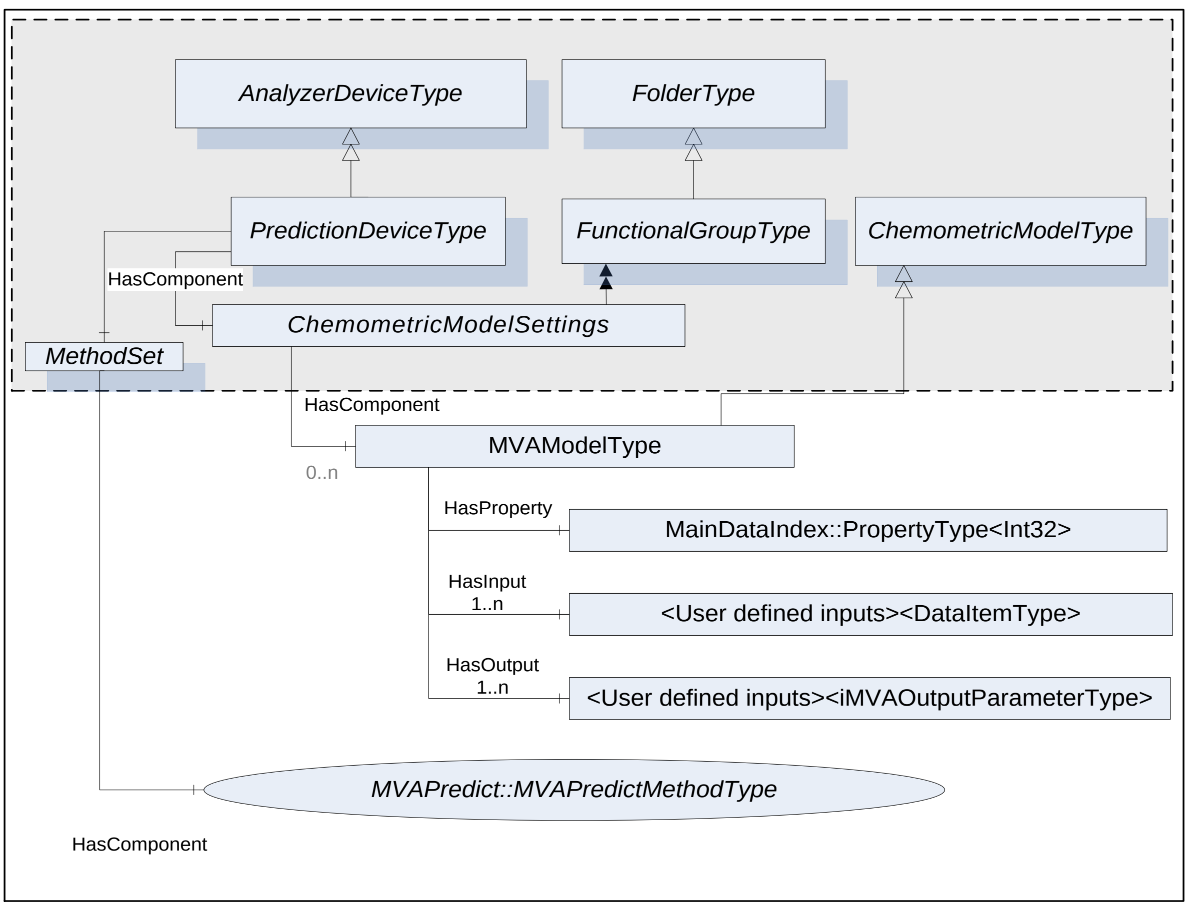

Figure 24 – MVAModelType

G.3 MVAModelType

The MVA*Model Variables* are used to hold the description of a mathematical process and associated information to convert scaled data into one or more derived values. MVAModelType is formally defined in Table 115

All MVA*Model Variables* are located in the *ChemometricModelSettings FunctionalGroup* on a *Stream.* It may also be located *ChemometricModelSettings FunctionalGroup* on the AnalyzerDevice if they are used only in the context of the prediction service.

Table 115 – MVAModelType Definition

Attribute	**Value**				
BrowseName	MVAModelType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the ChemometricModelType defined in ADI specification					
HasProperty	Variable	MainDataIndex	Int32	PropertyType	Mandatory
HasOutput	Variable	<User defined Output#>	-	MVAOutputParameterType	OptionalPlaceholder

MainDataIndex is the index of the Inputs parameter that is used as MainData for the source timestamp. All derived / predicted data will have this timestamp.

The output parameter descriptions, referred by HasOutput ordered references, shall appear in the same order as the Outputs array of the MVAPredict method. It shall be possible to use these parameters directly without having to do intermediate mathematic or method call.

Table 116 summarizes constraints on *Variable Attributes* for *MVAModelType*.

Table 116 - Setting OPC UA Variable Attributes for MVAModelType

Attributes/Properties	**Description**
Value	Binary blob containing all elements of the chemometric model
DataType	ByteString
ValueRank	Always set to -1 (Scalar)
ArrayDimensions	Not applicable

G.3.1 MVAOutputParameterType

The MVAOutputParameterType describes output paramaters of the MVAModelType and MVAPredictMethodType.

MVAOutputParameter*Type* is formally defined in Table 117.

Table 117 – MVAOutputParameterType Definition

Attribute	Value				
BrowseName	MVAOutputParameterType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the DataItemType defined in Part 8					
HasProperty	Variable	WarningLimits	Range	PropertyType	Optional
HasProperty	Variable	AlarmLimits	Range	PropertyType	Optional
HasProperty	Variable	AlarmState	AlarmStateEnumeration	PropertyType	Mandatory
HasProperty	Variable	VendorSpecificError	String	PropertyType	Optional
HasComponent	Variable	Statistics	MVAOutputParameterType []	BaseDataVariableType	OptionalPlaveholder

WarningLimits and AlarmLimits describe the ranges used to determine the acceptable limits of the resulting numerical MVAOutputParameter value. These values shall be set for numerical values.

In terms of automation, if value is:

value < AlarmLimits.Low → ALARM_LOW

WarningLimits.Low ≤ value < AlarmLimits.Low → WARNING_LOW

WarningLimits.Low ≤ value ≤ WarningLimits.High → NORMAL

WarningLimits.High < value ≤ AlarmLimits.High → WARNING_HIGH

AlarmLimits.High < value → ALARM_HIGH

AlarmState describes if the resulting MVAOutputParameter value is acceptable for example within the value limits. However, a value may be between the limits and still be in alarm due to other model consideration or for example, if a classification model is not able to classify a given sample.

Table 118 – AlarmStateEnumeration Values

Value	Description
NORMAL_0	Normal
WARNING_LOW_1	In low warning range
WARNING_HIGH_2	In high warning range
WARNING_4	In warning range (low or high) or some other warning cause
ALARM_LOW_8	In low alarm range
ALARM_HIGH_16	In high alarm range
ALARM_32	In alarm range (low or high) or some other alarm cause

The Statistics is an array of statistics generated at the same time as the MVAOutputParameter that qualifies it.

The VendorSpecificError contains detailed vendor specific error message explaining the alarm state.

The DataType attribute of MVAOutputParameter may be:

- AnalogItemType for scalar value or unstructured array. In this case, WarningLimits and AlarmLimits shall be set. EngineeringUnits should be set.
- ArrayItemType subtype if parameters like spectrum.
- DataItemType for String

G.3.1.1 Good practices for MVA input and output parameters

It is strongly recommended to express MVAModel parameters in terms of high level types, for example, a spectrometer produces spectra that are YArrayItemType. The address space should expose a single parameter for it rather than N scalar values where N is the number of data points in the spectrum. This may imply that models shall be built using some convention rules, but doing so, really simplify the interaction with the prediction service of the ADI server. For example:

- If all scalar variables defining an YArrayItemType are prefixed with something like "NIR_", then the SetConfiguration, LoadModel may easily detect it and create the right parameter type.
- Use a convention where the first variable is the MainData variable or prefixing the MainData variable variables with the "MainData_" prefix may help to automatically find the MainDataIndex.

The Predict method should be able to extract the required range from a high level type input parameter. For example, if the input parameter is a spectrum with a X axis range of 400cm^{-1} to 5000cm^{-1}, it shall be possible to pass a spectrum with a range of 200cm^{-1} to 6000cm^{-1} to the Predict method and the Predict method shall be able to extract the right region.

To guarantee the correctness of the predictions, the server should apply some validation rules to verify that the input parameters are compatible with the model, for example, for spectral data, the validation may include:

- The alignment of the sampling grid of spectral data shall be compatible with the model.
- The spectral range of the input spectrum is wide enough to cover the range expected by the model.

Inputs and Outputs parameters shall not be a brutal dump of the API of the vendor predictor, but rather express in terms of what an end user needs to see. It does not forbid exposing the API structures, but often these structures are very difficult to use for "process clients" like DCS or SCADA.

G.3.2 MVAPredictMethodType

All MVAModels may be predicted directly by calling the MVAPredict method described in Table 119.

Table 119 - MVAPredictMethodType

Method	**Description**			
MVAPredict	Predict Outputs using the TargetModel against these Inputs.			
	InputArguments			
	Name	**dataType**	**ValueRank / arrayDimension**	**Description**
	TargetModel	NodeId	-1[0]	NodeId of the MVAModel to be predicted.
	MainDataIndex	Int32	-1[0]	Index of the Inputs parameter that is used as MainData for the source timestamp.
	Inputs	ObjectType	1[x]	Input parameters as defined by Inputs.
	OutputArguments			
	Name	***dataType***	***arraySize/ arrayDimension***	***Description***
	Outputs	ObjectType	1[x]	Output parameters as defined by Outputs.

All fields of each Inputs and Outputs arguments must be filled, the user relies on them to discover the model information.

The vendor specific predictor return code should be returned using the DiagnosticInfo.SymbolicId.

The role of the MVAPredict method is really to allow a client to predict against the target MVAModel. The MVAPredict method:

- May be called at any time after the AnalyserDeviceStateMachine reaches the Operating state if the MVAModels are located in the ChemometricModelSettings located at AnalyserDevce level. Potential conflicts with AnalyserChannel operations like loading configuration shall be handled by the implementer following the rules describe below.
- May be called at any time after the AnalyserChannel reaches the Idle state, during Idle, Starting and Execute state, if the MVAModel are attached to a given Stream. This will avoid potential conflicts with AnalyserChannel operations like loading configuration.
- May be called more than once concurrently by the same client.
- May be called concurrently by more than one client at the same time.
- Shall produce output MVAOutputParameters that may be used directly without having to do intermediate mathematic or method call.

This is the server responsibility to deal with multi-threading issues. The client side shall never have to deal with these issues. It is perfectly legal to serialize all requests on a given model and even at the Predictor level, as long as the client does not have to be aware of it.
The MVAPredict method Executable and UserExecutable attributes shall be used to signal when MVAPredict may be called or not.

The implementer may allow some MVAModels to be updated directly by writing the Value attribute of the MVAModel. For example, let say that the Value attribute is an exact copy of the model file, the client application only has to use the OPC UA Write service to transfer the model directly to the UA server. If the model is too large to fit in a single UA message, the model may be transferred in chunk using the NumericRange parameter of the Write service.

All ADI time management rules apply:

- All output parameters shall have the same timestamp as the main parameter, the one defined by MainDataIndex.

To avoid unexpected behaviours, the following rules must be applied when models are updated with the Write service:

- ADI server shall verify the integrity / validity of the Model ByteString and returned an error code if not correct without modifying anything in the actual server configuration. If the model is transferred in chunks, the final verification shall be done on the write of the last chunk.
- ADI address shall be updated to reflect the new model.
- ADI server shall handle multi-threading issues where a GetConfiguration, SetConfiguration GetConfigDataDigest and others, while updating the models. This may be done by serializing the requests as long as the client does not have to be aware of it.
- ADI server shall return the vendor specific predictor return code using the DiagnosticInfo.SymbolicId.

If the MVAModel is used to evaluate values appearing in any AcquisitionData FunctionalGroup, the following supplemental rules must also be applied:

- MVAModel objects may only be updated when the AnalyserChannel is in Stop state.
- ADI server shall update/change the Configuration and the ConfigDataDigest if the new model is different from the previous one and only in that case.

Annex H (normative) Namespace and Mappings

This appendix defines the numeric identifiers for all of the numeric *NodeIds* defined in this standard. 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 the numeric value for the *NodeId*.

The *BrowsePath* for an *Instance Node* is constructed by appending the *BrowseName* of the instance *Node* to the *BrowseName* for the containing instance or type. An underscore character is used to separate each *BrowseName* in the path. Let's take for example, the *DeviceType ObjectType Node* which has the *SerialNumber Property*. The **Name** for the *SerialNumber InstanceDeclaration* within the *DeviceType* declaration is: *DeviceType_SerialNumber*.

The *NamespaceUri* for all *NodeIds* defined here is http://opcfoundation.org/UA/ADI/

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

http://www.opcfoundation.org/UA/ADI/1.1/NodeIds.csv

The latest CSV that is compatible with this version of the standard can be found here:

http://www.opcfoundation.org/UA/ADI/NodeIds.csv

The Information Model Schema released with this version of the standard can be found here:

http://www.opcfoundation.org/UA/ADI/1.1/Opc.Ua.Adi.NodeSet2.xml

The latest Information Model schema that is compatible with this version of the standard can be found here:

http://www.opcfoundation.org/UA/ADI/Opc.Ua.Adi.NodeSet2.xml