SIGMATEK OPC_UA Service

Date of creation: 26.08.2016 | Version date: 16.03.2023 | Article number: XX-XXX-XXX-E

Publisher: SIGMATEK GmbH & Co KG
A-5112 Lamprechtshausen
Tel.: +43/6274/4321
Fax: +43/6274/4321-18
Email: office@sigmatek.at
WWW.SIGMATEK-AUTOMATION.COM

Translation from German

Contents

1 OPC_UA Introduction

1.1 What is OPC_UA?

OPC Unified Architecture, short OPC_UA, is an industrial communication protocol.

As the newest of all OPC specifications of the OPC Foundation, OPC_UA differs considerably from its predecessors, mainly in its capability to not only transfer but also machine-readably semantically describe machine data (control variables, measurement values, parameters, etc.). A binary UA communication protocol based on TCP is used for data exchange. Additionally, other protocols like HTTP or HTTPS are supported.

The OPC_UA participants can be controls, master computers, ERP systems, and many others.

With the SIGMATEK OPC_UA class and the connected OPC_UA_Server or OPC_UA_Client class, the following data can be transferred without major programming effort:

- → Transfer of simple data types: DINT, UDINT, REAL and STRING
- → Transfer of complex data types: BYTESTRING and structures
- → OPC_UA client data transfer to external OPC_UA servers
- → File transfer from client to server and vice versa

The OPC_UA class only offers basic functions for this:

- → Basic functionality for connecting the server and / or client class
- → encryption
- → Logging function

1.2 Contents of Delivery

OPC_UA class and sticker

Article number OPC_UA Embedded License (in the form of license sticker) 02-010-074

1.3 Placement

The license sticker has to be applied next to the type label of the hardware, where the OPC_UA software is installed.

1.4 Supported OPC_UA Services

5.4 Discovery Service Set:	Find Server	N
	GetEndpoints	Y
5.5 SecureChannel Service Set:	OpenSecureChannel	Y
	CloseSecureChannel	Y
5.6 Session Service Set:	CreateSession	Y
	ActivateSession	Y
	CloseSession	Y
	Cancel	N
5.7. NodeManagement Service Set:	AddNodes	N
	AddReferences	N
	DeleteNodes	N
	DeleteReferences	N
5.8. View Service Set:	Browse	Y
	BrowseNext	Y
	TranslateBrowsePathsToNodeIds	Y
	RegisterNodes	N
	UnregisterNodes	N
5.9. Query Service Set:	QueryFirst	N
	QueryNext	N
5.10. Attribute Service Set	Read	Y
	HistoryRead	Y
	Write	Y
	HistoryUpdate	N
5.11. Method Service Set:	Call	Y
5.12 MonitoredItem Service Set:	CreateMonitoredItems	Y
	ModifyMonitoredItems	N
	SetMonitoringMode	N
	SetTriggering	N
	DeleteMonitoredItems	N
5.13 Subscription Service Set:	Create Subscription	Y
	ModifySubscription	N
	SetPublishingMode	N
	Publish	Y
	Republish	N
	TransferSubscriptions	N
	DeleteSubscriptions	Y

1.5 Supported OPC_UA Features and Profiles

1.5.1 General

- Standard UA Server

1.5.2 Data Access

- DataAccess Server Facet
- ComplexType Server Facet

1.5.3 Events

- Basic Event Subscription Server Facet
- Address Space Notifier Server Facet

1.5.4 Methods

- Method Server Facet

1.5.5 Alarms & Conditions

- A&C Simple Server Facet
- A&C Address Space Instance Server Facet
- A&C Enable Server Facet
- A&C Alarm Server Facet
- A&C Acknowledgeable Alarm Server Facet
- A&C Exclusive Alarming Server Facet
- A&C Non-Exclusive Alarming Server Facet

1.5.6 Historical Access

- Historical Raw Data Server Facet
- Historical Data AtTime Server Facet

1.6 Supported Datatypes

Table 1 lists the basic data types together with their value ranges. Fields with a gray background indicate data types that are currently not supported. IDs of the data types of other address spaces can be found in the corresponding documentation.

Table 1: IDs of the basic data types. Types with gray background are currently not supported.

ID	Name	Description
1	Boolean	A logical value with two states (true or false).
2	SByte	An integer value between -128 and 127 including.
3	Byte	An integer value between 0 and 255 including.
4	Int16	An integer value between -32 768 and 32 767 including.
5	UInt16	An integer value between 0 and 65 535 Including.
6	Int32	An integer value between -2 147 483 648 and 2 147 483 647 including.
7	UInt32	An integer value between 0 and 4 294 967 295 including.
8	Int64	An integer value between -9 223 372 036 854 775 808 and 9 223 372 036 854 775 807 including.
9	UInt64	An integer value between 0 and 18 446 744 073 709 551 615 including.
10	Float	An IEEE single precision (32 bit) floating point value.
11	Double	An IEEE double precision (64 bit) floating point value.
12	String	A sequence of Unicode characters.
13	DateTime	An instance in time.
14	Guid	A 16-byte value that can be used as a globally unique identifier.
15	ByteString	A sequence of octets (bytes).
16	XmlElement	An XML element.
17	NodeId	An identifier for a node in the address space of an OPC_UA Server.
18	ExpandedNodeId	A NodeId that allows the namespace URI to be specified instead of an index.
19	StatusCode	A numeric identifier for an error or condition associated with a value or operation.
20	QualifiedName	A name qualified by a namespace.
21	LocalizedText	Human readable text with an optional locale identifier.
22	ExtensionObject	A structure that contains an application-specific data type that may not be recognized by the receiver.

2 Server Data Transfer with external OPC_UA Clients

Up to version 4.xx the OPC_UA class also independently supported the simple server functionality (reading and writing of data points).

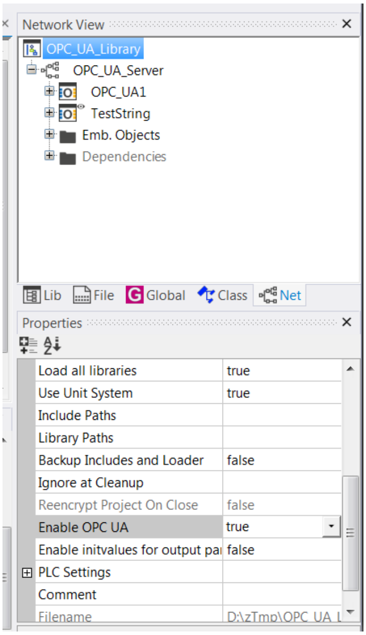
In November 2020, the server functions were expanded and during this they were moved to a new separate module (module name OPC_UA_Server).

From version 5.0 of the OPC_UA class, the **"OPC_UA_Server" module** must also be placed in the network and connected to the OPC_UA class for the server functions.

To implement an OPC_UA server communication in a LASAL project, the following steps must be followed:

→ In each station that should work as an OPC_UA server first the described OPC_UA class has to be imported and placed in a network. Furthermore, an OPC_UA_Server class must be placed and connected to the OPC_UA class.

→ Additionally, OPC_UA has to be enabled in the project!



Load all libraries	true
Use Unit System	true
Include Paths	
Library Paths	
Backup Includes and Loader	false
Ignore at Cleanup	
Reencrypt Project On Close	false
Enable OPC UA	true
Enable initvalues for output pa	false
PLC Settings	
Comment	
Filename	D:\zTmp\OPC_UA_L

2.1 Manual Start of the Service

If the steps described above are followed, the OPC_UA service starts automatically after the boot process of the control. Since this is not always desired, you also have the option of starting the service manually at a later point in the runtime.
To activate this, the second bit of the client "Config" is set to 1. The start command can then be set from the CmdManualStart server. There are two different starting situations:

A: Server "CmdManualStart" is initialized with "0":
The service is not yet started; however, the OPC_UA configuration XMLs are already loaded at startup so that a later start can take place more quickly.

B: Server "CmdManualStart" is initialized with "-1":
The XMLs are NOT loaded here. This saves memory if the service is not to be started at all in the current runtime period.

3 Client Data Transfer with External OPC_UA Servers

Up to version 2.6, the OPC_UA class also independently supported the simple client functionality (reading and writing of data points).

In June 2018 the client functions were extended and thereby outsourced in a new separate module (module name OPC_UA_Client).

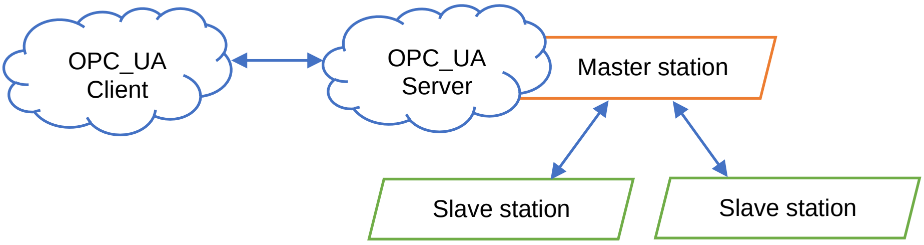
As of version 3.0 of the OPC_UA class, the **"OPC_UA_Client" module** must therefore also be placed in the network for the client functions and be connected to the OPC_UA_class

The old OPC_UA_Client.xml files are still supported.

4 MultiStation

The MultiStation feature is a part of OPC_UA that allows data transfer between the OPC_UA Server and slave stations connected in an Ethernet network.
The communication protocol based on the multi-master configuration is used for data exchange.
The configuration of the data points belonging to the master-slave communication is defined in an XML file.

Schematic overview of the MultiStation function:



It is completely transparent for the client from which PLC the values are actually taken. Existing clients therefore do not have to be changed.

4.1 Configuration

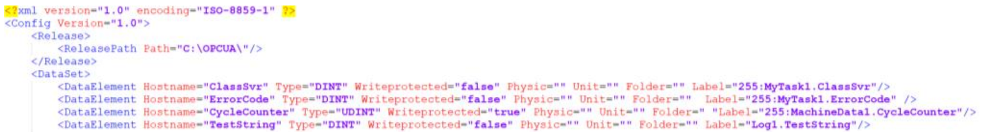
To add the OPC_UA endpoint to each slave station, it is necessary to include the station number in the label name.

Example: **`Label="255:MyTask1.ClassSvr"`**

An example of such a configuration is shown below.
For values located on the master station, the default label name is used without a number.

There can be a maximum of 256 slave stations.
The station number can be anything between -2,147,483,648 and 2,147,483,647.
However, it is highly recommended to use numbers between 0 and 255 as this may change in future versions.

Example of a configuration showing the extension of the label with the station number:



```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<Config Version="1.0">
    <Release>
        <ReleasePath Path="C:\OPCUA\"/>
    </Release>
    <DataSet>
        <DataElement Hostname="ClassSvr" Type="DINT" Writeprotected="false" Physic="" Unit="" Folder="" Label="255:MyTask1.ClassSvr"/>
        <DataElement Hostname="ErrorCode" Type="DINT" Writeprotected="false" Physic="" Unit="" Folder=""  Label="255:MyTask1.ErrorCode" />
        <DataElement Hostname="CycleCounter" Type="UDINT" Writeprotected="true" Physic="" Unit="" Folder="" "Label="255:MachineData1.CycleCounter"/>
        <DataElement Hostname="TestString" Type="DINT" Writeprotected="false" Physic="" Unit="" Folder="" Label="Log1.TestString"/>
```

4.2 Supported Features

MultiStation supports the following features:

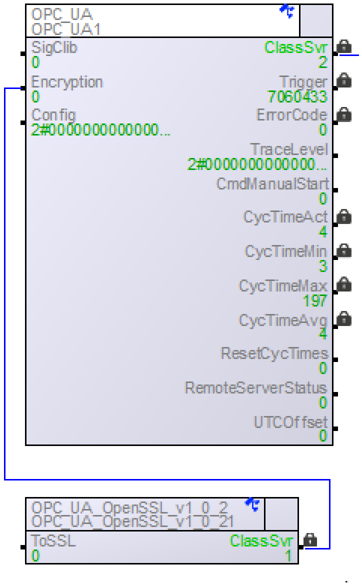
- Parsing of all data points to each master station
- Initialization of master and slave stations
- Sending numeric and string values from the OPC_UA client to the slave station
- Update the values of data points in the master station based on any change in the slave station
- TCP communication between master and slave station
- Adding servers to the update list
- Error handling

4.3 Supported Data Types

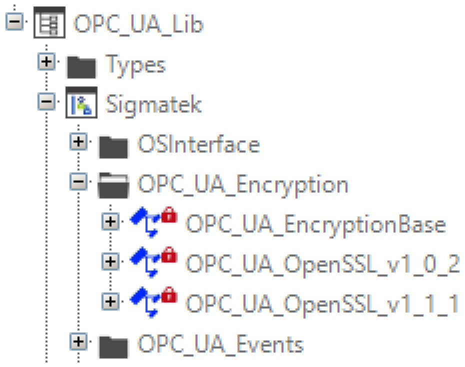
Possible data types to be transferred between the stations:

- DINT
- UDINT
- REAL
- STRING

5 OPC_UA Class



For the functionality of the OPC_UA service an instance of this class must be placed in a network.
Furthermore, one of the supplied derivatives of the OPC_UA_EncryptionBase class must be connected to the client Encryption.



CPUs with RTK	… OPC_UA_OpenSSL_v1_0_2
CPUs with Salamander	… OPC_UA_OpenSSL_v1_0_2
CPUs with Gecko >= V09.07.081	… OPC_UA_OpenSSL_v1_1_1

Important: Only one of the two classes may be in the project!
If both classes are in the same project, several linker errors will occur, and the project will not be executable.

On program start an own thread is created, in which the OPC_UA service runs. No further programming is necessary.

5.1 Interface Connections

5.1.1 Server

ClassSrv	progress for initialization (details are described below ...)	
	0	standard value at program start The "InitAllModules" method is called until all registered modules have reported "Ready". Then the "FunctSetUp" method is called. It is expected, that additional XML configuration files are provided if necessary ("AddXmlConfig" resp. "OPCUA_AddXmlConfig"). Finally the method InitAllProvider is called.
	1	Now the OPC_UA service is started. If the "Manual" start mode is required by configuration, the service is only started with the corresponding command.
	2	status after error free execution of the method "OPCUA_ServerStart" Here the method "OPCUA_CyclicRun" is called.
Trigger	Ia running counter increased with each internal cycle (). Returns a status, whether internal processing is active or not.	
ErrorCode	error codes for eventually occurring errors (details described below ...)	
	0	no errors
	-1	if the method "OPCUA_Init" has not been called as the first method
	-2	call without previous initialization - the "OPCUA_Init" method was not called
	-3	call of the method "OPCUA_AddXmlConfig" after starting the server - at this point in time, no more configuration changes are allowed
	-4	call of the method "OPCUA_ServerStart" without calling the method "OPCUA_AddXmlConfig" before
	-5	internal error when starting the OPC_UA service (see log files)
	-6	call of the method "OPCUA_CyclicRun" without calling the method "OPCUA_ServerStart" before
	-7	internal error during processing the OPC_UA protocol (see log files)
	-1001	configuration file does not exist
	-1002	length of the configuration file could not be determined
	-1003	contents of the configuration file could not be read
	-1004	reading configuration file failed (incorrect structure)
	-2001	configuration file for EM77 does not exist
	-2004	failed to read the user file (structure or entry incorrect)
	-3004	failed to read the EM77 file (structure or entry incorrect)

TraceLevel	Shows the current trace level (details described below ...) This server can also be written - so the TraceLevel can therewith be change via this server during runtime. The server is designed to be written to using checkboxes. The mechanism to write to the server is explained in the class *_Bit32*. The tracing entries are written to a file - name: **event16.log** (event16.bak) - this is located on the controller in the folder **C:\sysmsg\** The single levels are bit masks and so can be combined bitwise. From version 5.2 of the OPC_UA class the resetting of the TraceLevel is supported. By default, logging of ERROR and WARNING is assumed. If a TraceLevel is set that is higher (in the sense of the number of entries) than the two above, the TraceLevel is reset to the default value after a certain time that can be set via the SetParameter method (default 3600 seconds). It is irrelevant whether the value was set initially or during the program run. If a TraceLevel is initially set that is lower than the default value, it is reset to this value.	
	OPCUA_TRACE_LEVEL_CONTENT Data packet output (OPC_UA protocol), including content	0x01 Bit 0
	OPCUA_TRACE_LEVEL_DEBUG ... debug information via the internal process in the OPC_UA server	0x02 Bit 1
	OPCUA_TRACE_LEVEL_INFO ... expanded system information	0x04 Bit 2
	OPCUA_TRACE_LEVEL_SYSTEM ... infrequent system events (start, stop, connect, ...)	0x08 Bit 3
	OPCUA_TRACE_LEVEL_WARNING ... system warnings	0x10 Bit 4
	OPCUA_TRACE_LEVEL_ERROR ... serious error	0x20 Bit 5
CmdManualStart	START command, if the start mode is “Manual” (the start mode is configured in the client Config) See also chapter "Manual Start of the Service" within this documentation.	
CycTimeAct	Current cycle time in [µs]. This time indicates how long the processing of pending requests currently takes.	
CycTimeMin	Minimum cycle time [µs]	
CycTimeMax	Maximum cycle time [µs]	
CycTimeAvg	Average cycle time [µs]	
ResetCycTimes	command "Reset Cycle Times" Setting to #1 will reset the above cycle times	

RemoteServerStatus	Status of the OPC_UA remote server, which last changed its status.
UTCOffset	Input: current UTC Offset including Summertime [minutes]. Value can be changed by writing to this server.

5.1.2 Clients

SigClib	object channel to SigLib (connection is established automatically)
Encryption	command channel: required connection to a derivation of OPC_UA_EncryptionBase. e.g.: OPC_UA_OpenSSL_v1_0_2
config	Bit pattern for configuration Bit 0 IdentifierType 0 = "Numeric" / 1 = "String" Bit 1 Start Mode 0 = "Automatic" / 1 = "Manual via server-command" Bit 2 Session Lifetime Handling 0 = “handling enabled” / 1 = “handling disabled” Usually the session is terminated and the memory is freed in the event of an unwanted disconnection after the timeout has expired. If "Session Lifetime Handling" is disabled, the session memory is retained if the connection is lost unintentionally. This option should only be activated in exceptional cases if the external client requires it. Bit 3 Disable dedicated memory 0 = "dedicated memory enabled" / 1 = "dedicated memory disabled" The dedicated memory reserves memory on startup to speedup the handling of small memory allocations. If this is not necessary or not needed, this allocation can be deactivated, also to reduce start-up memory consumption. Dedicated memory is activated by default. Bit x Reserve

5.1.3 Global Methods

Background	The class has a background method so that it is available in the case of a derivation. The method does not have to be activated and has no further meaning for the function of the class.
Init	Initializing and creating the OPC_UA thread.
FunctStart	Called once while starting the OPC_UA server and signals the user that this service was started.
FunctRun	Called cyclically, if the service was started.
AfterProviderInitialize	The method is called when all providers have been initialized. The call is forwarded to all registered modules.
GetLasalId	reads the unique Lasal ID for a desired server IN label name of the server OUT retcode unique Lasal ID
SetValue32	sets the value of a signed 32-bit server IN lasalid unique Lasal ID IN value new value OUT retcode -1= general error 0= access denied 1= OK
SetValueU32	Equivalent to the "SetValue32" method for the data type UDINT
SetValueF32	Equivalent to the "SetValue32" method for the data type REAL
GetValue32	Reads the value of a signed 32-bit server IN pvalue the read value is written to this address IN lasalid unique Lasal ID OUT retcode -1= general error 1= OK
GetValueU32	Equivalent to the “GetValue32" method for the data type UDINT
GetValueF32	Equivalent to the “GetValue32" method for the data type REAL
SetString16	Sets the value of a server of the type STRING IN lasalid unique Lasal ID IN pstr pointer to the new value OUT retcode -1= general error 0= access denied 1= OK

GetString16	Reads the contents of a server of the type STRING IN pdst pointer to which the value should be written IN max_chrlength maximum length of the string IN lasalid unique Lasal ID OUT retcode -1= general error 1= OK
GetString16Crc	Returns the CRC value for the transferred Lasal ID IN lasalid unique Lasal ID OUT retcode CRC value
CB_activateDS **CB_prepaireDS**	The Callback (CB) “f_CB_activateDS” is called, if a client wants to transfer and activate a data set to the control. This callback is used in the control program as a trigger for reading and activating the desired settings data set. The "f_CB_ prepaireDS" callback is called when a client requests a data set. This callback is used in the control program as a trigger for providing desired settings data set. The methods are redirected to the connected OPC_UA_ModuleBase objects. IN pID unique ID of the data set IN pName name of the data set IN pPath path, where the data set is located OUT retcode 0= OK, otherwise error code
CB_alarmList	The OPC_UA server calls this function during initialization. With this method, the OPC_UA server requests the list of all active alarms. The method is redirected to the connected OPC_UA_ModuleBase objects. OUT retcode reserved for future tasks, is not evaluated
CB_fileSystem	This Callback is called, if a file changed. The method is redirected to the connected OPC_UA_ModuleBase objects. IN typ type of the file change (1= file new, 2= file deleted, 3 = file changed) IN pPath path incl. file name and extension of the file OUT retcode reserved for future tasks, is not evaluated

CB_GetStringArray	This Callback is called by OPC_UA-Server, if a string array must be read. The method is redirected to the connected OPC_UA_ModuleBase objects. IN: nodeId … Pointer to the node ID, for which the string array should be read (Struct of the type OPCUA_NodeId). The node ID can be determined for the check via UA expert or the nodeset XML. IN: list … Pointer to pointer to return the list with the string pointers of the single strings in the string array IN: listCount … Pointer to return the number of strings in the string array OUT: retcode … 0 = OK
CB_SetStringArray	This Callback Is called by OPC_UA-Server, if a string array has to be written. The method is redirected to the connected OPC_UA_ModuleBase objects. IN: nodeId … Pointer to the node ID, for which the string array should be written (Struct of the type OPCUA_NodeId). The node ID can be determined for the check via UA expert or the nodeset XML. IN: list … Pointer to the list with the string pointers of the single strings in the string array IN: listCount … Number of strings in the string array OUT: retcode … 0 = OK

SetParameter	Can be used to set process specific parameters IN ParaNr — parameter number IN Value — new value for the wanted parameter OUT retcode — 0 = Parameter successfully set -1 Parameter was not set (wrong value, ...) valid parameter: 0 = **OPC_UA_PAR_SET_DELAYTIME** ... delay (cycle time) for OPC_UA thread Value: new value in milliseconds (default 10 ms) 1 = **OPC_UA_PAR_SET_THREADPRIO** ... Priority for the OPC_UA thread Value: Priority 1 to 13; default: 9 (16=RealTime, 14=Cyclic, 10=Background) **Attention**: must be called in the **Init BEFORE** the **_FirstScan**! 2 = **OPC_UA_PAR_SET_AUTOCERTIFNR** ... Interface number for automatic certificate creation Value: 1 = IF1; 2 = IF2 **Attention**: must be called in the **Init BEFORE** the **_FirstScan**! 3 = **OPC_UA_PAR_SET_DEDICATED_MEMORY_ENABLED** The dedicated memory reserves memory on startup to speedup the handling of small memory allocations. If this is not necessary or not needed, this allocation can be deactivated, also to reduce start-up memory consumption. Dedicated memory is enabled by default. **Attention**: must be set as initial value. This cannot be changed later! 4 = **OPC_UA_PAR_SET_TRACELEVEL_RESET_TIME** Sets the time after which a changed TraceLevel is reset to the default value or to a lower value set initially. The value is given in seconds. If the value is set to 0, the TraceLevel is not automatically reset to the default value.
NewSystemTime	Method is called when an OPC_UA client tries to set a new system time. The method can be overloaded if the user is to evaluate the system time. If the method is not overloaded, the private method SetSystemTime is called and the system time is automatically transferred. IN highDateTime H-UDINT time stamp Unix time (seconds since 01.01.1970) IN lowDateTime L-UDINT time stamp Unix time (seconds since 01.01.1970) OUT retcode 0

RegisterModule	Used to register a module (called by the module that wants to register). IN pModule — this pointer of the module IN multipleAllowed — FALSE = only one module of this type is allowed TRUE .... several modules of this type are allowed OUT retcode — 0 ... OK / -1 ... Error
ValidateUser	This method is used to check the user and password. The default implementation returns the value 0 (0 = no validation performed - validation is performed using the standard configuration file "UserConfiguration.xml"). If the processing is to be done in LASAL, this virtual method must be overwritten. If username/password are valid, a value > 0 must be returned. (registration accepted). If username/password are invalid, a value < 0 must be returned. (registration denied). IN userName — pointer to the user name IN password — pointer to the password OUT retcode — see description above
SetTimeZoneOffset	Method for setting the offset between UTC time and the local time (time zone & daylight saving time). e.g. GE daylight saving time = +2 hours … UTC offset = -7200 seconds IN: timeZoneOffset — offset of UTC time to local time in seconds
GetTimeZoneOffset	Returns the current setting of the UTC offset to the local time (time zones & daylight saving time) e.g. GE daylight saving time = +2 hours … UTC offset = -7200 seconds OUT: timeZoneOffset offset of UTC time to local time in seconds
GetLasalIdVariable	Get the Lasal-ID of a Lasal-variable IN label — Name of the Lasal variable OUT retcode — 0 = OK
CB_CertificateLoaded	Callback to the application. Is called every time an application certificate has been activated, e.g. after startup or after creating a certificate via CreateApplCertificate(). IN IF_Number — Number of the interface IN ValidTo — Expiration date of the certificate (seconds since 01.01.1970). OUT retcode — 0 = OK

CB_ValidateCertificate	Callback to the application. Is called when a new certificate is to be validated. The response can be made using the CertificateValidationFeedback() method. IN certificateInfo — Pointer to the certificate info (see CreateApplCertificate) IN identity — Pointer to the identity info (see CreateApplCertificate) IN fileName — Pointer to the file name of the stored certificate file IN certificateData — Pointer to the certificate data IN certificateLength — Length of the certificate data OUT retcode — 0 = OK
RegisterClient	Method is used internally to register OPC_UA_Client modules.
RegisterServer	Method is used internally to register OPC_UA_Server modules.
CB_CreateSession	Callback to the application. Is called each time a Client is creating a session. IN pApplicationUri — Pointer to the ApplicationUri of the client information
CB_CloseSession	Callback to the application. Is called each time a Client is closing a session. IN pApplicationUri — Pointer to the ApplicationUri of the client information
SetValue	Function executing the internal writing procedure of a server (see SetValue32, SetValueF32, …) IN lasalid — unique Lasal ID IN value — new value OUT retcode — -1= general error 0= access denied 1= OK
OpcUaThread	OPC_UA services are processed in this thread. IN pthis — pointer to the own instance
RegisterProvider	Registers the providers of all modules OUT retcode — 0 ... OK / <> 0 ... RegisterProvider() incorrect
GetTrigger	returns the current value of the server "Trigger"
GetCycTimeMin	returns the current value of the server "CycTimeMin"
GetCycTimeMax	returns the current value of the server "CycTimeMax"
GetCycTimeAvg	returns the current value of the server "CycTimeAvg"
ResetCycleTimes	calling this method resets all cycle times

CreateApplCertificate	Method for creating an application certificate (for secure communication).
	
	IN IF_Number — Interface number (1 = IF1 / 2 = IF2)
	IN CertificateInfo — Certificate data
	IN Identity — Certificate issuer data
	
	OUT retcode — 0 = no error
	-99 = SSL not available
	-1 = bad input parameter
	x = internal error
	
	Example:
	CertificateInfo.sURI := "urn:10.10.16.61:OPC_UA Embedded Server";
	CertificateInfo.sIP := "10.10.16.61";
	CertificateInfo.sDNS := "";
	CertificateInfo.sEMail := "";
	CertificateInfo.validTime := 31536000; // [sec]
	
	Identity.sOrganization := "SIGMATEK GmbH & Co KG";
	Identity.sOrganizationUnit := "Library 1";
	Identity.sLocality := "Lamprechtshausen";
	Identity.sState := "Salzburg";
	Identity.sCountry := "AT"; // 2 letter code!
	Identity.sCommonName := "OPC_UA Embedded Server";
	Identity.sDomainComponent := "";

GetApplCertificateDetails	Is called by the application. - Method returns all information of the current certificate for the desired interface. Then the method FreeCertificateDetails() should be called! IN IF_Number — Number of the desired interface (1 = IF1 / 2 = IF2) IN certificateInfo — Pointer to pointer to return the certificate info IN identity — Pointer to pointer to return the identity info OUT retcode — 0 = OK
CertificateValidationFeedback	Called by the application to announce the result of the certification validation. If "certificateData" and "certificateLength" are entered, then these data are used to overwrite a possibly existing certificate file. "certificateData" and "certificateLength" have a higher priority than certificates from the memory. IN fileName — Pointer to the file name (string) IN certificateData — Pointer to the certificate data IN certificateLength — Length of the certificate data IN feedback — Feedback (Reject, Trust, Revoke or Delete) OUT retcode — 0 = OK

FreeCertificateDetails	Must be called by the application to free the internal memory allocated by calling GetApplCertificateDetails().If the Free method is not called, a memory leak occurs!
	
	IN certificateInfo Pointer to the certificate info
	IN identity Pointer to the identity info
	OUT retcode 0 = OK
	
	Example:
	`VAR`
	`CertificateInfo : OPC_UA::tOpcUa_PkiCertificateInfo;`
	`Identity : OPC_UA::tOpcUa_PkiIdentity;`
	`END_VAR`
	`RetCode := OPC_UA.GetApplCertificateDetails( IF_Number:=2, certificateInfo:=#CertificateInfoDetail, identity:=#IdentityDetail);`
	`RetCode := OPC_UA.FreeCertificateDetails( certificateInfo:=CertificateInfoDetail, identity:=IdentityDetail);`

SetCertificateRootPath	Is called by the application to define the root folder of the certificates (default "C:\OPC_UA"). IN path Pointer to the root path (string) OUT retcode 0 = OK
AddLogEntry	Used this method for adding a logbook entry. … For details see also class UserLogging method AddEntry().
RegisterInterfaceClass	Used this method for registrating the OPC_UA_Interface class. IN pInterface This-pointer of the OPC_UA_Interface object
TimeUnixToOpcua	Method to convert UNIX time to OPC_UA time. UNIX time … seconds since 01.01.1970 OPC_UA time … 100 nanoseconds since 01.01.1601 IN UnixTime UNIX time to be converted IN pOpcUaTime_H Pointer to return the HighValue of OPC_UA time IN pOpcUaTime_L Pointer to return the LowValue of OPC_UA time
TimeOpcuaToUnix	Method to convert OPC_UA time to UNIX time. UNIX time … seconds since 01.01.1970 OPC_UA time … 100 nanoseconds since 01.01.1601 IN OpcUaTime_H HighValue of OPC_UA time IN OpcUaTime_L LowValue of OPC_UA time OUT UnixTime Calculated UNIX time

AddXmlConfig	reads a new/additional configuration file. So additional elements in the OPC_UA address space are registered. IN dpne path + file name + ext. where the data set is located OUT retcode 0= OK otherwise negative error code
UpdateServerState	For internal use by the class OPC_UA_Server.
UpdateClientState	For internal use by the class OPC_UA_Client.
BubbleFree **BubbleRealoc** **BubbleMalloc**	For internal use for bubble management.
GetElementsList	Gets the list of data items intended for use with the MultiStation function. The number of elements in the list is returned by the function. The pointer to the list is stored in the specified pointer. The memory still belongs to OPC_UA and must not be changed or released! IN pList Pointer to the list with the data elements. OUT NoElements Number of elements in the list.
RegisterStationManager	Registers the StationManager for use with the MultiStation function. IN pStationManager The pointer to the station manager to register at the OPC_UA class. Only one station manager can be registered. A second call to this method will overwrite the pointer to the former object. OUT retcode The retcode is always 0.

5.1.4 Private Methods

OPC_UA	Constructor initializes the OPC_UA interface OUT ret_code ConfStates
GetValue	function executing the internal reading procedure of a server (see GetValue32, GetValueF32, …) IN pvalue pointer to the read value IN lasalid unique Lasal ID OUT retcode -1= general error 1= OK
Setvalue32Changed	With this method, changes (of type DINT) to the settings data can be sent to the OPC_UA server. Together will all other “Setvalue..Changes” functions, this call has to be executed threadsafe. IN change general properties of the parameter change IN oldValue value before the change IN newValue current value / value after the change OUT state 0 Using OPC_UA Event, OPC_UA clients can be informed of changes in the settings data.
SetvalueU32Changed	Equivalent to the "Setvalue32Changed" method for the data type "UDINT".
SetvalueF32Changed	Equivalent to the "Setvalue32Changed" method for the data type "REAL".
SetvalueStringChanged	Equivalent to the "Setvalue32Changed" method for the data type "CHAR". Together will all other “Setvalue..Changes” functions, this call has to be executed threadsafe. IN change general properties of the parameter change IN oldValue value before the change IN newValue current value / value after the change OUT state 0
SetvalueStringChangedUC	Complies to the method "SetvalueStringChanged". The difference is that the input parameter in this method is transferred as an array of 16-bit values. So any UniCode characters can be transferred. Together will all other “Setvalue..Changes” functions, this call has to be executed threadsafe.
InitAllModules	Method to initialize all OPC_UA modules
InitAllProvider	Method to initialize all OPC_UA providers

InitAlarmCallback	With this method, the OPC_UA server can provided with a callback function. The OPC_UA server calls this function during initialization. With this method, the OPC_UA server requests the list of all active alarms. IN f_CB_alarmList pointer to the Callback function OUT retcode 0
InitDatasetCallback	With this method, the OPC_UA server can be provided with two callback functions. IN f_CB_activateDS CallBack for activating settings data sets f_CB_prepaireDS Callback for providing settings data sets OUT retcode 0 The Callback (CB) "f_CB_activateDS" is called, if a client wants to transfer and activate a data set to the control. This callback is used in the control program as a trigger for reading and activating the desired settings data set. The "f_CB_ prepaireDS" callback is called when a client requests a data set. This callback is used in the control program as a trigger for providing desired settings data set.
InitDatasetWorkingPath	With this method, the default path for operations with settings data for runtime can be defined. This path is, in addition to paths from the configuration, valid for all file operations. If this path is set, it is used as the default path for file operations without path specifications. E.g.: If the DatasetWorkingPath was set to c:\datenset\, the file is stored in the c:\datensatz\text.txt" directory when UploadFile is called with the "test.txt" parameter. IN path default path specification OUT retcode 0
InitFileSystemCallback	With this method, the OPC_UA server can provided with a callback function. IN f_CB_fileSystem Callback for FileSystem changes OUT retcode 0 The "f_CB_fileSystem" callback is called when a client triggers a change in the file system with a function call. All file functions (Upload File, Download File, Activate Dataset, Prepare Dataset) for example, thereby trigger changes in the file system and subsequently call this callback function.
InitVersionId	With this method, the OPC_UA server can sent a unique ID (version number). This ID can be later used for customer and control-specific implementations. IN versionId unique identification of the control version OUT retcode 0

SetTraceLevel	You can use this method to change the trace level at runtime. IN traceLevel trace level to be used OUT retcode 0
SetSystemTime	By calling this method, the system time can be set (UTC). IN highDateTime H-UDINT time stamp Unix time (seconds since 01.01.1970) IN lowDateTime L-UDINT time stamp Unix time (seconds since 01.01.1970) OUT retcode 0