# SIGMATEK
# OPC_UA
# Service

Erstellungsdatum: 26.08.2016 | Versionsdatum: 16.03.2023 | Artikelnummer: XX-XXX-XXX-D

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



Originalsprache

# Inhalt

# 1 OPC_UA Einführung

## 1.1 Was ist OPC_UA?

OPC Unified Architecture, kurz OPC_UA, ist ein industrielles Kommunikationsprotokoll.

Als neueste aller OPC-Spezifikationen der OPC Foundation unterscheidet sich OPC_UA erheblich von seinen Vorgängern, insbesondere durch die Fähigkeit, Maschinendaten (Regelgrößen, Messwerte, Parameter usw.) nicht nur zu transportieren, sondern auch maschinenlesbar semantisch zu beschreiben. Für den Datenaustausch wird ein auf TCP basierendes, binäres UA Kommunikationsprotokoll verwendet. Außerdem werden noch weitere Protokolle wie HTTP oder HTTPS unterstützt.

Die OPC_UA Teilnehmer können Steuerungen, Leitrechner, ERP-Systeme uvm. sein.

Mit der SIGMATEK OPC_UA-Klasse und angeschlossener OPC_UA_Server bzw. OPC_UA_Client Klasse können ohne großen Programmieraufwand folgende Daten übertragen werden:

- → Übertragung einfacher Datentypen: DINT, UDINT, REAL und STRING
- → Übertragung komplexer Datentypen: BYTESTRING und Strukturen
- → OPC_UA Client-Datentransfer zu externen OPC_UA-Servern
- → File-Transfer vom Client zum Server und umgekehrt

Die OPC_UA-Klasse bietet dafür nur grundlegende Funktionen:

- → Basis-Funktionalitäten für den Anschluss von Server und/oder Client-Klasse
- → Verschlüsselung
- → Logging Funktion

## 1.2 Lieferumfang

OPC_UA Package und Aufkleber

Artikelnummer OPC_UA Embedded License (in Form von Lizenzaufkleber): 02-010-074

## 1.3 Anbringung

Die Anbringung des Lizenzaufklebers erfolgt neben dem Typenetikett der Hardware, auf welcher die OPC_UA Software installiert ist.

## 1.4 Unterstützte OPC_UA-Services

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

## 1.5 Unterstützte 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 Unterstützte Datentypen

In Tabelle 1 sind die Basisdatentypen samt ihrer Wertebereiche aufgelistet. Grau hinterlegte Felder kennzeichnen Datentypen, die aktuell nicht unterstützt werden. IDs der Datentypen anderer Adressräume sind der entsprechenden Dokumentation zu entnehmen.

Tabelle 1: IDs der Basisdatentypen. Grau hinterlegte Typen werden aktuell nicht unterstützt.

| ID | Name | Beschreibung |
|---|---|---|
| **1** | Boolean | Ein logischer Wert mit zwei Zuständen (wahr oder falsch). |
| **2** | SByte | Ein ganzzahliger Wert zwischen -128 und 127 einschließlich. |
| **3** | Byte | Ein ganzzahliger Wert zwischen 0 und 255 einschließlich. |
| **4** | Int16 | Ein ganzzahliger Wert zwischen -32 768 und 32 767 einschließlich. |
| **5** | UInt16 | Ein ganzzahliger Wert zwischen 0 und 65 535 einschließlich. |
| **6** | Int32 | Ein ganzzahliger Wert zwischen -2 147 483 648 und 2 147 483 647 einschließlich. |
| **7** | UInt32 | Ein ganzzahliger Wert zwischen 0 und 4 294 967 295 einschließlich. |
| **8** | Int64 | Ein ganzzahliger Wert zwischen -9 223 372 036 854 775 808 und 9 223 372 036 854 775 807 einschließlich. |
| **9** | UInt64 | Ein ganzzahliger Wert zwischen 0 und 18 446 744 073 709 551 615 einschließlich. |
| **10** | Float | Ein IEEE-Gleitkommawert mit einfacher Genauigkeit (32 Bit). |
| **11** | Double | Eine IEEE-Gleitkommazahl mit doppelter Genauigkeit (64 Bit). |
| **12** | String | Eine Folge von Unicode-Zeichen. |
| **13** | DateTime | Eine Instanz in der Zeit. |
| **14** | Guid | Ein 16-Byte-Wert, der als global eindeutiger Bezeichner verwendet werden kann. |
| **15** | ByteString | Eine Folge von Oktetten (Bytes). |
| **16** | XmlElement | Ein XML-Element. |
| **17** | NodeId | Eine Kennung für einen Knoten im Adressraum eines OPC_UA Servers. |
| **18** | ExpandedNodeId | Eine NodeId, die es ermöglicht, den Namespace-URI anstelle eines Indexes anzugeben. |
| **19** | StatusCode | Ein numerischer Bezeichner für einen Fehler oder eine Bedingung, der/die mit einem Wert oder einer Operation zusammenhängt. |
| **20** | QualifiedName | Ein Name, der durch einen Namespace qualifiziert ist. |
| **21** | LocalizedText | Von Menschen lesbarer Text mit einem optionalen Gebietsschema-Bezeichner. |
| **22** | ExtensionObject | Eine Struktur, die einen anwendungsspezifischen Datentyp enthält, der vom Empfänger möglicherweise nicht erkannt wird. |

# 2 Server-Datentransfer mit externen OPC_UA Clients

Bis zur Version 4.xx unterstützte die OPC_UA-Klasse selbständig auch die einfache Server-Funktionalität (Lesen und Schreiben von Datenpunkten).

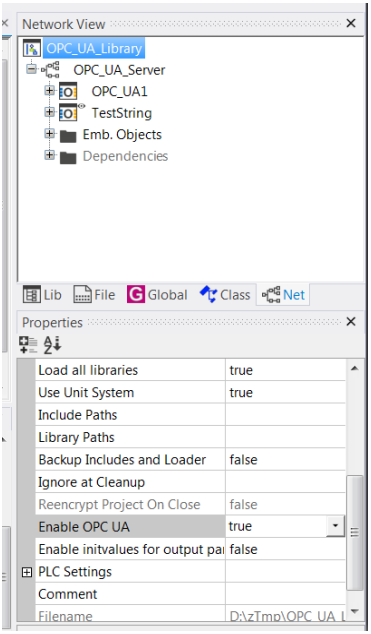
Im November 2020 wurden die Server-Funktionen erweitert und im Zuge dessen in ein neues separates Modul ausgelagert (Modulname OPC_UA_Server).

Ab der Version 5.0 der OPC_UA-Klasse muss somit für die Server-Funktionen zusätzlich das **Modul "OPC_UA_Server"** im Netzwerk platziert und an die OPC_UA-Klasse angeschlossen werden.

Um eine OPC_UA Server Kommunikation in einem LASAL Projekt zu realisieren, müssen folgende Schritte befolgt werden:

→ Bei jeder Station, die als OPC_UA Server arbeiten soll, muss die beschriebene OPC_UA Klasse importiert und in einem Netzwerk platziert werden.
Weiters ist eine OPC_UA_Server-Klasse zu platzieren und an die OPC_UA-Klasse anzuschließen.

→ OPC_UA muss wie im folgenden Screenshot gezeigt im Projekt aktiviert werden.



## 2.1 Manueller Start des Service

Werden die Schritte wie oben beschrieben befolgt, so startet der OPC_UA Service nach dem Bootvorgang der Steuerung automatisch. Nachdem dies nicht immer gewünscht ist, gibt es auch die Möglichkeit, den Service zu einem späteren Zeitpunkt der Laufzeit manuell zu starten.
Um dies zu aktivieren, wird das zweite Bit des Clients "Config" auf 1 gesetzt. Der Befehl zum Start kann dann über den Server "CmdManualStart" abgesetzt werden. Dabei gibt es zwei unterschiedliche Ausgangssituationen:

A: Server "CmdManualStart" wird mit "0" initialisiert:
Hierbei wird der Service noch nicht gestartet, damit ein späterer Start schneller erfolgen kann, werden aber die OPC_UA Konfigurations-XMLs bereits beim Hochfahren geladen.

B: Server "CmdManualStart" wird mit "-1" initialisiert:
Hier werden auch die XMLs NICHT geladen. Dies spart Speicher, wenn der Service in der aktuellen Laufzeitperiode gar nicht gestartet werden soll.

# 3 Client-Datentransfer mit externen OPC_UA Servern

Bis zur Version 2.6 unterstützte die OPC_UA-Klasse selbständig auch die einfache Client-Funktionalität (Lesen und Schreiben von Datenpunkten).

Im Juni 2018 wurden die Client-Funktionen erweitert und im Zuge dessen in ein neues separates Modul ausgelagert (Modulname OPC_UA_Client).

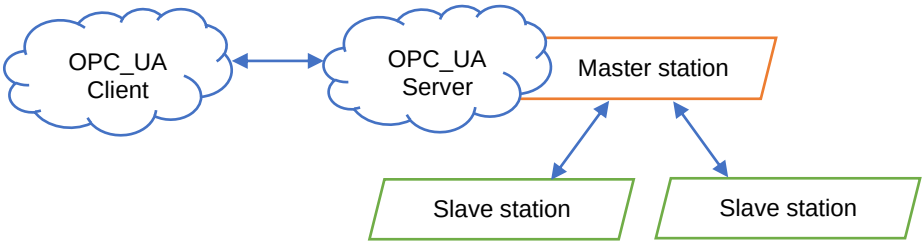
Ab der Version 3.0 der OPC_UA-Klasse muss somit für die Client-Funktionen zusätzlich das **Modul "OPC_UA_Client"** im Netzwerk platziert und an die OPC_UA_Klasse angeschlossen werden.

Die alten OPC_UA_Client.xml Dateien werden weiterhin unterstützt.

# 4 MultiStation

Die MultiStation-Funktion ist ein Teil von OPC_UA, der die Datenübertragung zwischen dem OPC_UA Server und den Slave-Stationen, die in einem Ethernet-Netzwerk verbunden sind, ermöglicht.
Das Kommunikationsprotokoll, das auf der Multi-Master-Konfiguration basiert, wird für den Datenaustausch verwendet.
Die Konfiguration der Datenpunkte, die zur Master-Slave-Kommunikation gehören, wird in einer XML-Datei definiert.

Schematischer Überblick über die MultiStation-Funktion:




Für den Client ist es völlig transparent, aus welcher SPS die Werte tatsächlich übernommen werden. Bestehende Clients müssen daher nicht geändert werden.

## 4.1 Konfiguration

Um den OPC_UA Endpunkt zu jeder Slave-Station hinzuzufügen, ist es notwendig, die Nummer der Station in den Namen des Labels einzufügen.

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

Ein Beispiel für eine solche Konfiguration ist nachfolgend dargestellt.
Für Werte, die sich auf der Master-Station befinden, wird der Standard-Label-Name ohne eine Nummer verwendet.

Es können maximal 256 Slave-Stationen vorhanden sein.
Die Nummer der Station kann beliebig zwischen -2.147.483.648 und 2.147.483.647 liegen.
Es wird jedoch dringend empfohlen, Nummern zwischen 0 und 255 zu verwenden, da dies in zukünftigen Versionen geändert werden kann.

Beispiel einer Konfiguration, das die Erweiterung des Labels mit der Stationsnummer zeigt:

```
<?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 Unterstützte Funktionen

MultiStation unterstützt die folgenden Funktionen:

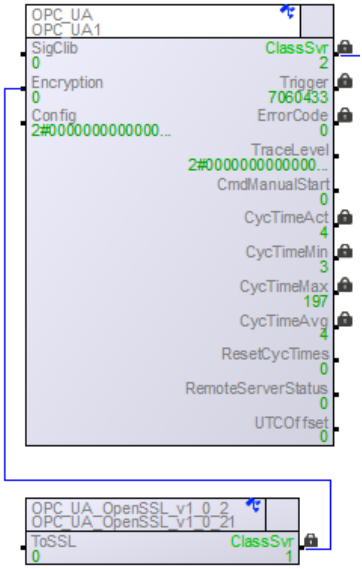
- -- Parsing aller Datenpunkte zu jeder Master-Station
- -- Initialisierung von Master- und Slave-Stationen
- -- Senden von numerischen und String-Werten vom OPC_UA Client zur Slave-Station
- -- Aktualisierung der Werte von Datenpunkten in der Master-Station, basierend auf jeder Änderung in der Slave-Station
- -- TCP-Kommunikation zwischen Master- und Slave-Station
- -- Hinzufügen von Servern zur Aktualisierungsliste
- -- Fehlerbehandlung

## 4.3 Unterstützte Datentypen

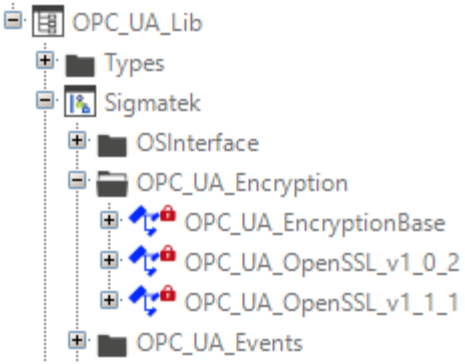
Mögliche Datentypen, die zwischen den Stationen übertragen werden sollen:

- -- DINT
- -- UDINT
- -- REAL
- -- STRING

# 5 OPC_UA Class




Für die Funktionalität des OPC_UA-Services muss eine Instanz dieser Klasse in einem Netzwerk platziert werden.
Weiters muss eine der mitgelieferten Ableitungen der Klasse OPC_UA_EncryptionBase an dem Client Encryption angeschlossen werden.



| | |
|---|---|
| CPUs mit RTK | … OPC_UA_OpenSSL_v1_0_2 |
| CPUs mit Salamander | … OPC_UA_OpenSSL_v1_0_2 |
| CPUs mit Gecko >= V09.07.081 | … OPC_UA_OpenSSL_v1_1_1 |

**Wichtig: Es darf nur eine der beiden Klassen im Projekt sein!**
Wenn beide Klassen im gleichen Projekt sind, kommt es zu mehreren Linker-Errors und das Projekt ist nicht lauffähig.

Beim Programmstart wird ein eigener Thread angelegt, in welchem der OPC_UA-Service läuft. Es sind hierfür keine weiteren programmtechnischen Tätigkeiten zu erledigen.

## 5.1 Schnittstellen

### 5.1.1 Server

| | | |
|---|---|---|
| **ClassSrv** | Fortschritt bei der Initialisierung (Details nachfolgend beschrieben …) | |
| | 0 | Standardwert bei Programmstart. Es wird die Methode "InitAllModules" solange aufgerufen, bis alle registrierten Module "Ready" gemeldet haben. Anschliessend wird die Methode "FunctSetUp" aufgerufen. Es wird erwartet, dass bei Bedarf weitere XML-Konfigurationsdateien zur Verfügung gestellt werden ("AddXmlConfig" bzw. "OPCUA_AddXmlConfig").<br>Abschließend wird die Methode InitAllProvider aufgerufen. |
| | 1 | Nun wird der OPC_UA Service gestartet. Wenn per Konfiguration die Startart "Manuell" gefordert ist, dann wird der Service erst mit dem dazugehörigen Befehl gestartet. |
| | 2 | Status nach fehlerfreier Ausführung der Methode "OPCUA_ServerStart". Hier wird die Methode "OPCUA_CyclicRun" aufgerufen. |
| **Trigger** | Ist ein laufender Zähler der bei jedem internen Zyklus () erhöht wird. Liefert somit einen Status, ob die interne Bearbeitung aktiv ist oder nicht. | |
| **ErrorCode** | Error-Code bei eventuell vorliegenden Fehlern (Details nachfolgend beschrieben…) | |
| | 0 | kein Fehler |
| | -1 | wenn die Methode "OPCUA_Init" nicht als erste Methode aufgerufen wurde |
| | -2 | Aufruf ohne vorherige Initialisierung.<br>- Die Methode "OPCUA_Init" wurde noch nicht aufgerufen. |
| | -3 | Aufruf der Methode "OPCUA_AddXmlConfig" nach dem Start des Servers.<br>- zu diesem Zeitpunkt sind keine Änderungen an der Konfiguration mehr erlaubt. |
| | -4 | Aufruf der Methode "OPCUA_ServerStart" ohne vorherigen Aufruf der Methode "OPCUA_AddXmlConfig" |
| | -5 | Interner Fehler beim Starten des OPCUA-Service (siehe Logfiles). |
| | -6 | Aufruf der Methode "OPCUA_CyclicRun" ohne vorherigen Aufruf der Methode "OPCUA_ServerStart". |
| | -7 | Interner Fehler während der Abarbeitung des OPC_UA Protokolls (siehe Logfiles). |
| | -1001 | Konfigurationsdatei nicht vorhanden |
| | -1002 | Länge der Konfigurationsdatei konnte nicht bestimmt werden |
| | -1003 | Inhalt der Konfigurationsdatei konnte nicht gelesen werden. |
| | -1004 | Lesen der Konfigurationsdatei fehlgeschlagen (fehlerhafte Struktur). |
| | -2001 | Konfigurationsdatei für EM77 nicht vorhanden |
| | -2004 | Lesen der Users-Datei fehlgeschlagen (Struktur oder Eintrag fehlerhaft). |
| | -3004 | Lesen der EM77-Datei fehlgeschlagen (Struktur oder Eintrag fehlerhaft). |

<table>
<tr><td>TraceLevel</td><td colspan="2">Zeigt den aktuellen Trace Level an (Details nachfolgend beschrieben …)<br><br>Durch Schreiben auf diesen Server kann der Trace Level auch zur Laufzeit geändert werden. Der Server ist konzipiert, um über Checkboxen beschrieben zu werden. Der Mechanismus dieses Schreibens ist in der Klasse _Bit32 erklärt.<br><br>Die Tracing Einträge werden in eine Datei geschrieben - Name: event16.log (event16.bak) - diese befindet sich auf der Steuerung im Ordner C:\sysmsg\<br><br>Die einzelnen Level verstehen sich als Bitmaske und können somit bitweise miteinander kombiniert werden.<br><br>Ab Version 5.2 der Klasse OPC_UA wird das Zurücksetzen des TraceLevel’s unterstützt. Standardmäßig wird das Loggen von ERROR und WARNING angenommen. Wird ein TraceLevel gesetzt, der höher (im Sinne der Menge der Einträge) ist als die beiden oben genannten, wird nach einer gewissen Zeit, die über die Methode SetParameter eingestellt werden kann (default 3600 Sekunden) das TraceLevel auf den Standardwert zurückgesetzt. Dafür ist es unerheblich, ob der Wert initial oder im Programmablauf gesetzt wurde. Wird initial ein TraceLevel gesetzt, das niedriger ist als der Standardwert, wird auf diesen Wert zurückgesetzt.</td></tr>
<tr><td></td><td>OPCUA_TRACE_LEVEL_CONTENT<br>… Ausgabe aller Datenpakete (OPC_UA Protokoll) inklusive Inhalt</td><td>0x01<br>Bit 0</td></tr>
<tr><td></td><td>OPCUA_TRACE_LEVEL_DEBUG<br>… Debug Informationen über den internen Ablauf im OPC_UA Server</td><td>0x02<br>Bit 1</td></tr>
<tr><td></td><td>OPCUA_TRACE_LEVEL_INFO<br>… Erweiterte Systeminformationen</td><td>0x04<br>Bit 2</td></tr>
<tr><td></td><td>OPCUA_TRACE_LEVEL_SYSTEM<br>… seltene Systemereignisse (start, stop, connect, …)</td><td>0x08<br>Bit 3</td></tr>
<tr><td></td><td>OPCUA_TRACE_LEVEL_WARNING<br>… Systemwarnungen</td><td>0x10<br>Bit 4</td></tr>
<tr><td></td><td>OPCUA_TRACE_LEVEL_ERROR<br>… schwerwiegende Fehler</td><td>0x20<br>Bit 5</td></tr>
<tr><td>CmdManualStart</td><td colspan="2">START-Befehl, wenn der Start Mode "Manuell" ist<br>(der Start Mode wird mittels des Config-Client konfiguriert)<br>Siehe auch Kapitel "Manueller Start des Service" innerhalb dieser Doku.</td></tr>
<tr><td>CycTimeAct</td><td colspan="2">Die aktuelle Zykluszeit in [µs]. Diese Zeit gibt an, wie lange die Abarbeitung der anstehenden Anfragen aktuell dauert.</td></tr>
<tr><td>CycTimeMin</td><td colspan="2">Zykluszeit Minimum [µs]</td></tr>
<tr><td>CycTimeMax</td><td colspan="2">Zykluszeit Maximum [µs]</td></tr>
<tr><td>CycTimeAvg</td><td colspan="2">Zykluszeit Mittelwert [µs]</td></tr>
<tr><td>ResetCycTimes</td><td colspan="2">Befehl "Reset Cycle Times"<br>Das Setzen auf #1 führt zum Reset der oben genannten Zykluszeiten</td></tr>
</table>

| | |
|---|---|
| **RemoteServerStatus** | Aktueller Zustand des OPC_UA RemoteServers, welcher als letzter seinen Zustand geändert hat. |
| **UTCOffset** | Eingabe: aktueller UTC-Offset einschliesslich der Sommerzeit [Minuten].<br>Der Wert kann durch Schreiben auf diesen Server geändert werden. |

### 5.1.2 Clients

| | |
|---|---|
| **SigClib** | Objekt Kanal zur SigClib (Verbindung wird automatisch gezogen) |
| **Encryption** | Kommando Kanal: Notwendige Verbindung zu einer Ableitung von OPC_UA_EncryptionBase.<br>z.B.: OPC_UA_OpenSSL_v1_0_2 |
| **config** | Bitmuster für Konfiguration<br>Bit 0 IdentifierType<br>0 = "Numeric" / 1 = "String"<br>Bit 1 Start Mode<br>0 = "Automatic" / 1 = "Manual via server-command"<br>Bit 2 Session Lifetime Handling<br>0 = “handling enabled” / “1” = handling disabled”<br>Normalerweise wird beim ungewollten Verbindungsabbruch nach Ablauf des Timeouts die Session beendet und der Speicher freigegeben.<br>Wenn das „Session Lifetime Handling“ disabled wird, dann bleibt beim ungewollten Verbindungsabbruch der Speicher der Session erhalten. Diese Option sollte nur in Ausnahmefällen eingeschaltet werden, wenn der externe Client dies erfordert.<br>Bit 3 Disable dedicated memory<br>0 = "dedicated memory enabled" / 1 = "dedicated memory disabled"<br>Der dedizierte Speicher wird beim Starten reserviert und bedient ausschließlich Anforderungen von kleinen Speichermengen, um deren Anforderung zu beschleunigen. Falls dies nicht notwendig oder gewünscht ist, kann dies deaktiviert werden. Außerdem sinkt durch die Deaktivierung der Speicherverbrauch beim Starten.<br>Dedizierter Speicher ist per default aktiviert.<br>Bit x Reserve |

### 5.1.3 Globale Methoden

| | |
|---|---|
| **Background** | Die Klasse besitzt eine Background-Methode, damit diese im Falle einer Ableitung zur Verfügung steht. Die Methode muss nicht aktiviert werden und hat für die Funktion der Klasse keine weitere Bedeutung. |
| **Init** | Initialisierungen und Anlegen des OPC_UA-Threads |
| **FunctStart** | Wird beim Starten des OPC_UA-Servers einmalig aufgerufen und signalisiert dem Anwender, dass dieser Dienst erfolgreich gestartet wurde. |
| **FunctRun** | Wird zyklisch aufgerufen, sofern der Dienst gestartet wurde. |
| **AfterProviderInitialize** | Die Methode wird aufgerufen, wenn alle Provider initialisiert wurden. Der Aufruf wird in alle registrierten Module weiter geleitet. |
| **GetLasalId** | Liest die eindeutige Lasal-ID für einen gewünschten Server<br><br>IN label — Name des Servers<br>OUT retcode — eindeutige LASAL-ID |
| **SetValue32** | Setzt den Wert eines signed 32-bit Servers<br><br>IN lasalid — eindeutige LASAL-ID<br>IN value — neuer Wert<br>OUT retcode — -1= allgemeiner Fehler<br>0= Zugriff verweigert<br>1= OK |
| **SetValueU32** | Äquivalent zur Methode "SetValue32" für den Datentyp UDINT |
| **SetValueF32** | Äquivalent zur Methode "SetValue32" für den Datentyp REAL |
| **GetValue32** | Liest den Wert eines signed 32-bit Servers<br><br>IN pvalue — Der gelesene Wert wird auf diese Adresse geschrieben<br>IN lasalid — eindeutige LASAL-ID<br>OUT retcode — -1= allgemeiner Fehler<br>1= OK |
| **GetValueU32** | Äquivalent zur Methode "GetValue32" für den Datentyp UDINT |
| **GetValueF32** | Äquivalent zur Methode "GetValue32" für den Datentyp REAL |
| **SetString16** | Setzt den Wert eines Servers vom Typ STRING<br><br>IN lasalid — eindeutige LASAL-ID<br>IN pstr — Zeiger auf den neuen Wert<br>OUT retcode — -1= allgemeiner Fehler<br>0= Zugriff verweigert<br>1= OK |

| | |
|---|---|
| **GetString16** | Liest den Inhalt eines Servers vom Typ STRING<br><br>IN pdst — Pointer auf den der Wert geschrieben werden soll<br>IN max_chrlength — maximale Länge des Strings<br>IN lasalid — eindeutige LASAL-ID<br>OUT retcode — -1= allgemeiner Fehler, 1= OK |
| **GetString16Crc** | Liefert den CRC Wert für die übergebene LASAL-ID<br><br>IN lasalid — eindeutige LASAL-ID<br>OUT retcode — CRC-Wert |
| **CB_activateDS**<br>**CB_prepaireDS** | Der Callback (CB) "f_CB_activateDS" wird aufgerufen, wenn ein Client einen Datensatz an die Steuerung übertragen und aktivieren will. Dieser Callback dient dem Steuerungsprogramm als Anstoß für das Einlesen und die Aktivierung des gewünschten Einstelldatensatzes.<br><br>Der Callback "f_CB_ prepaireDS " wird aufgerufen, wenn ein Client einen Datensatz anfordert. Dieser Callback dient dem Steuerungsprogramm als Anstoß für die Bereitstellung des gewünschten Einstelldatensatzes.<br><br>Die Methoden werden an die angeschlossenen OPC_UA_ModuleBase-Objekte weitergeleitet.<br><br>IN pID — eindeutige ID des Datensatzes<br>IN pName — Name des Datensatzes<br>IN pPath — Pfad, in dem sich der Datensatz befindet<br>OUT retcode — 0 = OK, ansonsten Fehlercode |
| **CB_alarmList** | Diese Callback Funktion wird vom OPC_UA Sever während der Initialisierung aufgerufen. Über diese Methode fordert der OPC_UA Server die Übertragung der Liste aller aktiven Alarme an.<br><br>Die Methode wird an die angeschlossenen OPC_UA_ModuleBase-Objekte weitergeleitet.<br><br>OUT retcode — Reserviert für zukünftige Aufgaben, wird nicht ausgewertet. |
| **CB_fileSystem** | Dieser Callback wird aufgerufen, wenn sich eine Datei geändert hat.<br><br>Die Methode wird an die angeschlossenen OPC_UA_ModuleBase-Objekte weitergeleitet.<br><br>IN typ — Typ der Dateiänderung (1 = Datei neu, 2 = Datei gelöscht, 3 = Datei geändert)<br>IN pPath — Pfad inkl. Dateiname und Extension der Datei<br>OUT retcode — Reserviert für zukünftige Aufgaben, wird nicht ausgewertet. |

| | |
|---|---|
| **CB_GetStringArray** | Dieser Callback wird vom OPC_UA-Server aufgerufen, wenn ein String-Array auszulesen ist.<br><br>Die Methode wird an die angeschlossenen OPC_UA_ModuleBase-Objekte weitergeleitet.<br><br>IN: nodeId … Pointer auf die Node-Id, für welche der String-Array ausgelesen werden soll (Struct vom Typ OPCUA_NodeId). Die Node-Id kann für die Überprüfung via UA-Expert oder der Nodeset-XML eruiert werden.<br>IN: list … Pointer auf Pointer zum Retournieren der Liste mit den String-Pointern der einzelnen Strings im String-Array<br>IN: listCount … Pointer zum Retournieren der Anzahl von Strings im String-Array<br>OUT: retcode … 0 = OK |
| **CB_SetStringArray** | Dieser Callback wird vom OPC_UA-Server aufgerufen, wenn ein String-Array zu schreiben ist.<br><br>Die Methode wird an die angeschlossenen OPC_UA_ModuleBase-Objekte weitergeleitet.<br><br>IN: nodeId … Pointer auf die Node-Id, für welche der String-Array geschrieben werden soll (Struct vom Typ OPCUA_NodeId). Die Node-Id kann für die Überprüfung via UA-Expert oder der Nodeset-XML eruiert werden.<br>IN: list … Pointer auf die Liste mit den String-Pointern der einzelnen Strings im String-Array<br>IN: listCount … Anzahl von Strings im String-Array<br>OUT: retcode … 0 = OK |

| | |
|---|---|
| **SetParameter** | Kann verwendet werden, um ablaufspezifische Parameter zu setzen.<br><br>IN ParaNr Parameternummer<br>IN Value neuer Wert für den gewählten Parameter<br>OUT retcode 0 = Parameter erfolgreich gesetzt<br>-1 = Parameter wurde nicht gesetzt (falscher Wert, …)<br><br>Gültige Parameter:<br>0 = **OPC_UA_PAR_SET_DELAYTIME** ... Delay (Zykluszeit) für OPC_UA Thread.<br>Value: neuer Wert in Millisekunden (default 10 ms)<br><br>1 = **OPC_UA_PAR_SET_THREADPRIO** ... Priorität für den OPC_UA Thread<br>Value: Priorität 1 bis 13; default: 9 (16=RealTime, 14=Cyclic, 10=Background)<br>**Achtung**: muss hierfür im **Init VOR** dem **_FirstScan** aufgerufen werden!<br><br>2 = **OPC_UA_PAR_SET_AUTOCERTIFNR**<br>... Interface Nummer für automatische Zertifikatserstellung<br>Value: 1 = IF1; 2 = IF2<br>**Achtung**: muss hierfür im **Init VOR** dem **_FirstScan** aufgerufen werden!<br><br>3 = **OPC_UA_PAR_SET_DEDICATED_MEMORY_ENABLED**<br>Der dedizierte Speicher wird beim Starten reserviert und bedient ausschließlich Anforderungen von kleinen Speichermengen, um deren Anforderung zu beschleunigen. Falls dies nicht notwendig oder gewünscht ist, kann dies deaktiviert werden. Außerdem sinkt durch die Deaktivierung der Speicherverbrauch beim Starten.<br>Dedizierter Speicher ist per default aktiviert.<br>**Achtung**: muss als Initialwert gesetzt sein. Der Wert kann später nicht mehr geändert werden!<br><br>4 = **OPC_UA_PAR_SET_TRACELEVEL_RESET_TIME**<br>Setzt die Zeit, nach der ein geändertes TraceLevel auf den Standardwert oder einen initial gesetzten niedrigeren Wert zurückgesetzt wird. Der Wert ist in Sekunden angegeben. Wird der Wert auf 0 gesetzt, wird das TraceLevel nicht automatisch auf den Standardwert zurückgesetzt. |
| **NewSystemTime** | Methode wird aufgerufen, wenn ein OPC_UA Client versucht, eine neue Systemzeit zu setzen. Die Methode kann überladen werden, wenn die Auswertung der Systemzeit vom Benutzer erfolgen soll. Wird die Methode nicht überladen, wird die Private Methode SetSystemTime aufgerufen und die Systemzeit automatisch übernommen.<br><br>IN highDateTime H-UDINT Zeitstempel Unixtime (Sekunden seit 01.01.1970)<br>IN lowDateTime L- UDINT Zeitstempel Unixtime (Sekunden seit 01.01.1970)<br>OUT retcode 0 |

| | |
|---|---|
| **RegisterModule** | Dient dem Registrieren eines Modules (wird vom entsprechenden Modul aufgerufen, welches sich registrieren möchte).<br><br>IN pModule this pointer des Modules<br>IN multipleAllowed FALSE = nur ein Modul dieses Typs ist erlaubt<br>TRUE = mehrere Module dieses Typs sind erlaubt<br>OUT retcode 0 ... OK / -1 ... Error |
| **ValidateUser** | Diese Methode wird benutzt, um den Benutzer samt Passwort zu überprüfen.<br>Die Standardimplementierung liefert den Wert 0 zurück<br>(0 = keine Validierung durchgeführt - Validierung wird mittels der standardmäßigen Konfigurationsdatei "UserConfiguration.xml" getätigt).<br><br>Soll die Bearbeitung in LASAL gemacht werden, so muss diese virtuelle Methode überschrieben werden.<br>Sind Benutzername/Passwort gültig, so muss ein Wert > 0 zurückgegeben werden<br>(Anmeldung zugelassen).<br>Sind Benutzername/Passwort ungültig, so muss ein Wert < 0 zurückgegeben werden<br>(Anmeldung verweigert).<br><br>IN userName Pointer auf Benutzername als String<br>IN password Pointer auf Passwort als String<br>OUT retcode siehe Beschreibung oben |
| **SetTimeZoneOffset** | Methode zum Setzen des Offsets zwischen der UTC-Zeit und der Lokalzeit (Zeitzone & Sommerzeit).<br>z.B. DE Sommerzeit = +2 Stunden … UTC-Offset = -7200 Sekunden<br><br>IN: timeZoneOffset Offset der UTC-Zeit zur Lokalzeit in Sekunden |
| **GetTimeZoneOffset** | Liefert die aktuelle Einstellung des UTC-Offset zur Lokalzeit (Zeitzone & Sommerzeit).<br>z.B. DE Sommerzeit = +2 Stunden … UTC-Offset = -7200 Sekunden<br><br>OUT: timeZoneOffset Offset der UTC-Zeit zur Lokalzeit in Sekunden |
| **GetLasalIdVariable** | Liefert die Lasal-ID einer Lasal-Variable<br><br>IN label Name der Lasal Variable<br>OUT retcode 0 = OK |
| **CB_CertificateLoaded** | Callback an die Applikation. Wird jedesmal aufgerufen, wenn ein Anwendungszertifikat aktiviert wurde, z.B. nach dem Hochfahren oder nach dem Anlegen eines Zertifikates via CreateApplCertificate().<br><br>IN IF_Number Nummer der Schnittstelle<br>IN ValidTo Ablaufdatum des Zertifikats (Sekunden seit 01.01.1970).<br>OUT retcode |

| | |
|---|---|
| **CB_ValidateCertificate** | Callback an die Applikation. Wird aufgerufen, wenn ein neues Zertifikat zu validieren ist.<br>Die Antwort kann mittels der Methode CertificateValidationFeedback() erfolgen.<br><br>IN certificateInfo Pointer auf die Zertifikats-Infos (siehe CreateApplCertificate)<br>IN identity Pointer auf die Identity-Infos (siehe CreateApplCertificate)<br>IN fileName Pointer auf File-Namen der abgelegten Zertifikats-Datei<br>IN certificateData Pointer auf Zertifikats-Daten<br>IN certificateLength Länge der Zertifikats-Daten<br>OUT retcode 0 = OK |
| **RegisterClient** | Methode wird intern verwendet zum Registrieren von OPC_UA_Client-Modulen. |
| **RegisterServer** | Methode wird intern verwendet zum Registrieren von OPC_UA_Server-Modulen. |
| **CB_CreateSession** | Callback an die Applikation.<br>Wird jedes Mal aufgerufen, wenn ein Client eine Sitzung erstellt.<br><br>IN pApplicationUri Pointer to the ApplicationUri of the client information |
| **CB_CloseSession** | Callback an die Applikation.<br>Wird jedes Mal aufgerufen, wenn ein Client eine Sitzung schliesst.<br><br>IN pApplicationUri Pointer to the ApplicationUri of the client information |
| **SetValue** | Funktion welche den internen Schreibvorgang eines Servers durchführt (siehe SetValue32, SetValueF32, …)<br><br>IN lasalid eindeutige Lasal-ID<br>IN value neuer Wert<br>OUT retcode -1= allgemeiner Fehler<br>0= Zugriff verweigert<br>1= OK |
| **OpcUaThread** | OPC_UA Dienste werden in diesem Thread abgearbeitet.<br><br>IN pthis Zeiger auf die eigene Instanz |
| **RegisterProvider** | Registriert die Provider von allen Modulen<br><br>OUT retcode 0 ... OK / <> 0 ... RegisterProvider() fehlerhaft |
| **GetTrigger** | retourniert den aktuellen Wert des Servers "Trigger" |
| **GetCycTimeMin** | retourniert den aktuellen Wert des Servers "CycTimeMin" |
| **GetCycTimeMax** | retourniert den aktuellen Wert des Servers "CycTimeMax" |
| **GetCycTimeAvg** | retourniert den aktuellen Wert des Servers "CycTimeAvg" |
| **ResetCycleTimes** | Der Aufruf dieser Methode führt zum Reset aller Zykluszeiten. |

| | |
|---|---|
| **CreateApplCertificate** | Methode zum Erstellen eines Applikations-Zertifikates (für gesicherte Kommunikation).<br><br>IN IF_Number Nummer der Schnittstelle (1 = IF1 / 2 = IF2)<br>IN CertificateInfo Daten des Zertifikats<br>IN Identity Daten des Zertifikat-Ausstellers<br><br>OUT retcode 0 = no error<br>-99 = SSL not available<br>-1 = bad input parameter<br>x = internal error<br><br>Beispiel:<br>CertificateInfo.sURI := "urn:10.10.16.61:OPC_UA Embedded Server";<br>CertificateInfo.sIP := "10.10.16.61";<br>CertificateInfo.sDNS := "";<br>CertificateInfo.sEMail := "";<br>CertificateInfo.validTime := 31536000; // [sec]<br><br>Identity.sOrganization := "SIGMATEK GmbH & Co KG";<br>Identity.sOrganizationUnit := "Library 1";<br>Identity.sLocality := "Lamprechtshausen";<br>Identity.sState := "Salzburg";<br>Identity.sCountry := "AT"; // 2 letter code!<br>Identity.sCommonName := "OPC_UA Embedded Server";<br>Identity.sDomainComponent := ""; |
| **GetApplCertificateDetails** | Wird seitens Applikation aufgerufen - Methode liefert alle Infos des aktuellen Zertifikates für die gewünschte Schnittstelle. Anschließend sollte die Methode FreeCertificateDetails() aufgerufen werden!<br><br>IN IF_Number Nummer der gewünschten Schnittstelle (1 = IF1 / 2 = IF2)<br>IN certificateInfo Pointer auf Pointer zum Retournieren der Zertifikats-Infos<br>IN identity Pointer auf Pointer zum Retournieren der Identity-Infos<br>OUT retcode 0 = OK |
| **CertificateValidationFeedback** | Wird seitens Applikation aufgerufen zum Bekanntgeben des Ergebnisses der Zertifizierungsvalidierung.<br>Wenn "certificateData" und "certificateLength" eingetragen sind, dann werden diese Daten verwendet, um eine eventuell vorhandene Zertifikatdatei zu überschreiben. "certificateData" und "certificateLength" haben eine höhere Priorität als Zertifikate aus dem Speicher.<br><br>IN fileName Pointer auf den File-Namen (String)<br>IN certificateData Pointer auf die Zertifikats-Daten<br>IN certificateLength Länge der Zertifikats-Daten<br>IN feedback Feedback (Reject, Trust, Revoke oder Delete)<br>OUT retcode 0 = OK |

| | |
|---|---|
| **FreeCertificateDetails** | Muss seitens Applikation aufgerufen werden zur Freigabe des internen Speichers, welcher durch den Aufruf von GetApplCertificateDetails() allokiert wurde. Wird die Free-Methode nicht aufgerufen, so entsteht ein Memory-Leak!<br><br>IN certificateInfo Pointer auf die Zertifikats-Infos<br>IN identity Pointer auf die Identity-Infos<br>OUT retcode 0 = OK<br><br>Beispiel:<br>VAR<br>CertificateInfo : OPC_UA::tOpcUa_PkiCertificateInfo;<br>Identity : OPC_UA::tOpcUa_PkiIdentity;<br>END_VAR<br>RetCode := OPC_UA.GetApplCertificateDetails( IF_Number:=2, certificateInfo:=#CertificateInfoDetail, identity:=#IdentityDetail);<br>RetCode := OPC_UA.FreeCertificateDetails( certificateInfo:=CertificateInfoDetail, identity:=IdentityDetail); |
| **SetCertificateRootPath** | Wird seitens Applikation aufgerufen um den Stammordner der Zertifikate festzulegen (default "C:\OPC_UA").<br><br>IN path Pointer auf den Root-Pfad (String)<br>OUT retcode 0 = OK |
| **AddLogEntry** | Diese Methode dient zum Hinzufügen eines Logbucheintrags.<br>… Weitere Informationen siehe auch Klasse UserLogging, Methode AddEntry(). |
| **RegisterInterfaceClass** | Diese Methode dient zum Registrieren der OPC_UA-Schnittstellenklasse.<br><br>IN pInterface This-pointer des OPC_UA_Interface Objektes |
| **TimeUnixToOpcua** | Methode zum Konvertieren der UNIX-Time in die OPC_UA-Time.<br>UNIX-Time … Sekunden seit 01.01.1970<br>OPC_UA-Time … 100 Nanosekunden seit 01.01.1601<br><br>IN UnixTime UNIX-Zeit, welche konvertiert werden soll<br>IN pOpcUaTime_H Pointer zum Retournieren des HighValue der OPC_UA-Time<br>IN pOpcUaTime_L Pointer zum Retournieren des LowValue der OPC_UA-Time |
| **TimeOpcuaToUnix** | Methode zum Konvertieren der OPC_UA-Time in die UNIX-Time.<br>UNIX-Time … Sekunden seit 01.01.1970<br>OPC_UA-Time … 100 Nanosekunden seit 01.01.1601<br><br>IN OpcUaTime_H HighValue der OPC_UA-Time<br>IN OpcUaTime_L LowValue der OPC_UA-Time<br>OUT UnixTime errechnete UNIX-Zeit |

| | |
|---|---|
| **AddXmlConfig** | Liest eine neue/zusätzliche Konfigurationsdatei ein. Somit werden zusätzliche Elemente im OPC_UA Adressraum registriert.<br><br>IN dpne — Pfad + Dateiname + Ext. in dem sich Datensatz befindet.<br>OUT retcode — 0= OK<br>ansonsten negativer Fehlercode |
| **UpdateServerState** | Zur internen Verwendung durch die Klasse OPC_UA_Server. |
| **UpdateClientState** | Zur internen Verwendung durch die Klasse OPC_UA_Client. |
| **BubbbleFree**<br>**BubbbleRealoc**<br>**BubbleMalloc** | Zur internen Verwendung für das Bubble-Management. |
| **GetElementsList** | Ermittelt die Liste der Datenelemente, die für die Verwendung mit der MultiStation-Funktion vorgesehen sind. Die Anzahl der Elemente in der Liste wird von der Funktion zurückgegeben. Der Zeiger auf die Liste wird in dem angegebenen Zeiger gespeichert. Der Speicher gehört weiterhin OPC_UA und darf nicht verändert oder freigegeben werden!<br><br>IN pList — Zeiger auf die Liste mit den Datenelementen<br>OUT NoElements — Anzahl der Elemente in der List. |
| **RegisterStationManager** | Registriert den StationManager für die Verwendung mit der MultiStation-Funktion.<br><br>IN pStationManager — Der Zeiger auf den Stationsmanager, der in der OPC_UA-Klasse registriert werden soll. Es kann nur ein Stationsmanager registriert werden. Ein zweiter Aufruf dieser Methode überschreibt den Zeiger auf das vorherige Objekt.<br>OUT retcode — Der retcode ist immer 0. |

### 5.1.4 Private Methoden

| | |
|---|---|
| **OPC_UA** | Konstruktor. Initialisiert die OPC_UA Schnittstelle.<br><br>OUT ret_code ConfStates |
| **GetValue** | Funktion welche den internen Lesevorgang eines Servers durchführt (siehe GetValue32, GetValueF32, …)<br><br>IN pvalue Zeiger auf den gelesenen Wert<br>IN lasalid eindeutige Lasal-ID<br>OUT retcode -1= allgemeiner Fehler<br>1= OK |
| **Setvalue32Changed** | Mit dieser Methode können Änderungen (vom Typ "DINT") am Einstelldatensatz an den OPC_UA Server übergeben werden.<br>Dieser Aufruf muss zusammen mit allen anderen „Setvalue..Changed“ Funktionen threadsicher durchgeführt werden.<br><br>IN change Allgemeine Eigenschaften der Parameteränderung<br>IN oldValue Wert vor der Änderung<br>IN newValue Aktueller Wert / Wert nach der Änderung<br>OUT state 0<br><br>OPC_UA Clients können sich mittels OPC_UA Event über Änderungen am Einstelldatensatz benachrichtigen lassen. |
| **SetvalueU32Changed** | Äquivalent zur Methode "Setvalue32Changed" für den Datentyp UDINT |
| **SetvalueF32Changed** | Äquivalent zur Methode "Setvalue32Changed" für den Datentyp REAL |
| **SetvalueStringChanged** | Äquivalent zur Methode "Setvalue32Changed" für den Datentyp CHAR<br>Dieser Aufruf muss zusammen mit allen anderen „Setvalue..Changed“ Funktionen threadsicher durchgeführt werden.<br><br>IN change Allgemeine Eigenschaften der Parameteränderung<br>IN oldValue Wert vor der Änderung<br>IN newValue Aktueller Wert / Wert nach der Änderung<br>OUT state 0 |
| **SetvalueStringChangedUC** | Entspricht der Methode "SetvalueStringChanged". Der Unterschied liegt darin, dass der Übergabeparameter bei dieser Methode als Array von 16-bit Werten übergeben wird. Somit können beliebige UniCode Zeichen übertragen werden.<br>Dieser Aufruf muss zusammen mit allen anderen „Setvalue..Changed“ Funktionen threadsicher durchgeführt werden. |
| **InitAllModules** | Methode zum Initialisieren aller OPC_UA Module |
| **InitAllProvider** | Methode zum Initialisieren aller OPC_UA Provider |

| | |
|---|---|
| **InitAlarmCallback** | Mit dieser Methode kann dem OPC_UA Server eine Callback Funktion bereitgestellt werden. Diese Callback Funktion wird vom OPC_UA Server während der Initialisierung aufgerufen. Über diese Methode fordert der OPC_UA Server die Übertragung der Liste aller aktiven Alarme an.<br><br>IN f_CB_alarmList Zeiger auf die Callback Funktion<br>OUT retcode 0 |
| **InitDatasetCallback** | Mit dieser Methode können dem OPC_UA Server zwei Callback Funktionen bereitgestellt werden.<br><br>IN f_CB_activateDS CallBack für Aktivierung von Einstelldatensätzen<br>IN f_CB_prepaireDS CallBack für Bereitstellung von Einstelldatensätzen<br>OUT retcode 0<br><br>Der Callback (CB) "f_CB_activateDS" wird aufgerufen, wenn ein Client einen Datensatz an die Steuerung übertragen und aktivieren will. Dieser Callback dient dem Steuerungsprogramm als Anstoss für das Einlesen und die Aktivierung des gewünschten Einstelldatensatzes.<br><br>Der Callback "f_CB_ prepaireDS" wird aufgerufen, wenn ein Client einen Datensatz anfordert. Dieser Callback dient dem Steuerungsprogramm als Anstoss für die Bereitstellung des gewünschten Einstelldatensatzes. |
| **InitDatasetWorkingPath** | Mit dieser Methode kann der Standardpfad für Operationen mit dem Einstelldatensatz zur Laufzeit festgelegt werden. Dieser Pfad ist neben den freigegebenen Pfaden aus der Konfiguration für alle Dateioperationen gültig. Ist dieser Pfad gesetzt, wird er als Standardpfad für Dateioperationen ohne Pfadangaben verwendet. Z.B: Wenn der DatasetWorkingPath auf "c:\datensatz\" festgelegt wurde, dann wird bei einem Aufruf von UploadFile mit Parameter "test.txt", die Datei im Verzeichnis "c:\datensatz\text.txt" gespeichert.<br><br>IN path Angabe des Standardpfades<br>OUT retcode 0 |
| **InitFileSystemCallback** | Mit dieser Methode kann dem OPC_UA Server eine Callback Funktion bereitgestellt werden.<br><br>IN f_CB_fileSystem Callback für Änderungen am File System<br>OUT retcode 0<br><br>Der Callback "f_CB_fileSystem" wird aufgerufen, wenn ein Client durch einen Funktionsaufruf eine Veränderung am Dateisystem verursacht. So verursachen z.B. alle Dateifunktionen (Upload File, Download File, Activate Dataset, Prepare Dataset) Änderungen am Dateisystem und haben somit einen Aufruf dieser Callback Funktion zur Folge. |
| **InitVersionId** | Mit dieser Methode kann dem OPC_UA Server eine eindeutige Id (Versionsnummer) übergeben werden. Diese Id kann zukünftig für kunden- bzw. steuerungsspezifische Implementierungen verwendet werden.<br><br>IN versionId eindeutige Kennung der Steuerungsversion<br>OUT retcode 0 |

| | |
|---|---|
| **SetTraceLevel** | Mit dieser Methode kann der Trace Level zur Laufzeit verändert werden.<br><br>IN traceLevel Trace Level der verwendet werden soll<br>OUT retcode 0 |
| **SetSystemTime** | Mit dem Aufruf dieser Methode kann Die Systemzeit gesetzt werden (UTC).<br><br>IN highDateTime H-UDINT Zeitstempel Unixtime (Sekunden seit 01.01.1970)<br>IN lowDateTime L- UDINT Zeitstempel Unixtime (Sekunden seit 01.01.1970)<br>OUT retcode 0 |