



BrbLibUa V1.03

Dokumentation

B&R übernimmt keine Haftung für Folgen, die durch die Implementierung sowie die Benutzung dieser Software entstehen!

Inhaltliche Änderungen dieses Dokuments behalten wir uns ohne Ankündigung vor. B&R haftet nicht für technische oder drucktechnische Fehler und Mängel in diesem Dokument. Außerdem übernimmt B&R keine Haftung für Schäden, die direkt oder indirekt auf Lieferung, Leistung und Nutzung dieses Materials zurückzuführen sind. Wir weisen darauf hin, dass die in diesem Dokument verwendeten Soft- und Hardwarebezeichnungen und Markennamen der jeweiligen Firmen dem allgemeinen warenzeichen-, marken- oder patentrechtlichen Schutz unterliegen.



Inhaltsverzeichnis

Inhaltsverzeichnis.....	2
1 Allgemeines	5
1.1 Hinweise zum Compiler	5
1.2 Abhängigkeiten	5
1.3 Hinweise zu StructuredText und anderen IEC-Sprachen	5
1.4 Getestet mit UnitTests	5
1.5 Geprüft mit ClangTidy	5
1.6 Neueste Versionen auf GitHub	6
2 Revisionsgeschichte	7
2.1 BrbLibUa V1.03 – 2024-02-22	7
2.1.1 Hinweise zum Compiler	7
2.1.2 Code-Prüfung mit ClangTidy	7
2.1.3 Versionen geändert.....	7
2.1.4 Abhängigkeit geändert	7
3 Fehlernummern	8
3.1 eBRB_ERR_UA_ERROR = 51000	8
3.2 eBRB_ERR_UA_NO_ELEMENTS = 51001	8
3.3 eBRB_ERR_UA_INVALID_INDEX = 51002	8
3.4 eBRB_ERR_UA_NOT_CONNECTED = 51003	8
3.5 eBRB_ERR_UA_NOT_RUNNING = 51004.....	8
4 Pakete	9
4.1 General	9
4.1.1 Status.....	9
4.1.1.1 BrbUaGetStatusCodeText.....	9
4.1.2 Nodes.....	9
4.1.2.1 BrbUaSetNodeId	9
4.1.2.2 BrbUaSetNodeIdNum.....	10
4.1.2.3 BrbUaGetRandomNodeId	10
4.1.2.4 BrbUaAreNodeIdsEqual	10
4.1.2.5 BrbUaAddNodeIdText	11
4.1.3 Attributes.....	11
4.1.3.1 BrbUaGetAttributeIdDatatype.....	11
4.1.3.2 BrbUaGetAttributeList	12
4.1.3.3 BrbUaAddBooleanText.....	14
4.1.3.4 BrbUaAddNodeClassText	14
4.1.3.5 BrbUaAddDatatypeIdText	14
4.1.3.6 BrbUaAddArrayDimensionText	15
4.1.3.7 BrbUaAddAccessLevelText.....	15
4.1.3.8 BrbUaAddEventNotifierText	16
4.1.4 VariantValues.....	16
4.1.4.1 Datentyp.....	16
4.1.4.2 BrbUaClearVariantValue	17
4.1.4.3 BrbUaConvVariantValueFromString.....	17
4.1.4.4 BrbUaConvVariantValueToString.....	18
4.1.4.5 BrbUaAddVariantValueSubName	19
4.1.5 ByteStrings.....	19
4.1.5.1 Grundsätzliches zum Datentyp ByteString	19
4.1.5.2 Datentyp UAByteStringBrbEventId_TYP	20
4.1.5.3 Umwandlung eines ByteStrings in Hex-Darstellung	20

4.1.5.4 BrbUaIncByteString	20
4.1.5.5 BrbUaGetRandomByteString	20
4.1.6 Images	21
4.1.6.1 Datentypen	21
4.1.6.2 BrbUaLoadImage	22
4.1.7 LocalizedTexts	22
4.1.7.1 BrbUaSetLocalizedText	22
4.1.7.2 BrbUaGetRandomLocalizedText	23
4.1.7.3 BrbUaAddLocalizedTextText	23
4.1.8 QualifiedNames	24
4.1.8.1 BrbUaSetQualifiedName	24
4.1.8.2 BrbUaGetRandomQualifiedName	24
4.1.9 TimeAndDate	24
4.1.9.1 BrbDtStructToUaSrvDateTime	24
4.1.9.2 BrbUaSrvDateTimeToDtStruct	25
4.1.9.3 BrbUaGetSrvTimeText	25
4.1.9.4 BrbUaGetRandomTimezone	26
4.1.10 ServerInfo	26
4.1.10.1 Namespace-Array	26
4.1.10.2 ServerStatus	26
4.1.11 ServerDiag	27
4.1.11.1 ServerDiagData	27
4.1.11.2 SessionDiagData	28
4.1.11.3 SessionSecurityData	29
4.1.11.4 SubscriptionDiagData	31
4.1.11.5 Hinweise zu den Diagnose-Daten	32
4.1.11.5.1 Ermitteln der Client-Anzahl	32
4.1.11.5.2 Ermitteln der Clients	32
4.1.11.5.3 Aktive und inaktive Clients	32
4.1.12 Additional	33
4.1.12.1 BrbUaGetRandomGuidIdString	33
4.1.12.2 BrbUaGetRandomXmlElement	33
4.2 Client	33
4.2.1 Connection	33
4.2.1.1 BrbUaGetConnectionStatusText	34
4.2.2 RunClient	34
4.2.2.1 Allgemeines	34
4.2.2.2 Performance und Speicher-Verbrauch	35
4.2.2.3 Datenobjekt	35
4.2.2.3.1 Allgemeines	35
4.2.2.3.2 Connection	36
4.2.2.3.3 Namespaces	37
4.2.2.3.4 NodeHandles	37
4.2.2.3.5 ReadBlocks	38
4.2.2.3.6 WriteBlocks	38
4.2.2.3.7 Methods	38
4.2.2.3.8 Subscription	39
4.2.2.4 Funktionsbausteine zum Betrieb des RunClients	41
4.2.2.4.1 BrbUaRunClientInit	41
4.2.2.4.2 BrbUaRunClientCyclic	42
4.2.2.4.3 BrbUaRunClientExit	43
4.2.2.5 RunClient-Struktur	43
4.2.2.5.1 Cfg	44
4.2.2.5.2 eCmd	44
4.2.2.5.3 Connection	45
4.2.2.5.4 Namespaces/NodeHandles/ReadBlocks/WriteBlocks/Methods/Subscriptions	45
4.2.2.5.5 State	46
4.2.2.6 Ausführen von vordefinierten Funktionen	47
4.2.2.6.1 BrbUaRcReadBlock	47
4.2.2.6.2 BrbUaRcWriteBlock	48
4.2.2.6.3 BrbUaRcCallMethod	48
4.2.2.7 Zugriff auf die internen Daten	49

4.2.2.7.1 BrbUaRcGetSrvNamespace	49
4.2.2.7.2 BrbUaRcGetNodeHandle	49
4.2.2.7.3 BrbUaRcGetReadBlock.....	50
4.2.2.7.4 BrbUaRcGetReadItem	50
4.2.2.7.5 BrbUaRcGetWriteBlock.....	51
4.2.2.7.6 BrbUaRcGetWriteItem.....	51
4.2.2.7.7 BrbUaRcGetMethod	52
4.2.2.7.8 BrbUaRcGetArgument	53
4.2.2.7.9 BrbUaRcGetSubscription	53
4.2.2.7.10 BrbUaRcSetSubscription.....	54
4.2.2.7.11 BrbUaRcGetMonitoredItem	55
4.2.2.7.12 BrbUaRcSetMonitoredItem	56
4.2.2.7.13 BrbUaRcGetMiValueChanged	57
4.2.2.7.14 BrbUaRcGetMiRemainingValueCount	58
4.2.2.7.15 BrbUaRcGetEventItem	58
4.2.2.7.16 BrbUaRcGetEventItemReceiveCount	59
4.2.2.7.17 BrbUaRcGetEventItemReceived	59
4.2.2.7.18 BrbUaRcGetEventField	60
4.2.2.8 BrbUaRcMonitor.....	61
4.2.2.9 Tipps zur Implementierung, Inbetriebnahme und Fehlersuche	61
4.2.2.9.1 Beispiel für ein MonitoredItem	62
4.2.2.9.2 ReadBlocks, WriteBlocks und Methods.....	63
4.3 Server.....	65
4.3.1 Methods	65
4.3.1.1 BrbUaSrvHandleMethod	65
4.3.1.1.1 Struktur.....	65
4.3.1.1.2 BrbUaSrvHandleMethod	66
4.3.1.2 BrbUaGetMethodOperateActionText	66
4.3.2 RunServer.....	67
4.3.2.1 Allgemeines.....	67
4.3.2.2 Performance und Speicher-Verbrauch	67
4.3.2.3 Datenobjekt	68
4.3.2.3.1 Allgemeines.....	68
4.3.2.3.2 Namespaces	68
4.3.2.3.3 Events + EventFields	69
4.3.2.4 Funktionsbausteine zum Betrieb des RunServers	69
4.3.2.4.1 BrbUaRunServerInit	69
4.3.2.4.2 BrbUaRunServerCyclic	70
4.3.2.4.3 BrbUaRunServerExit.....	70
4.3.2.5 RunServer-Struktur	71
4.3.2.5.1 Cfg.....	71
4.3.2.5.2 Namespaces/Events	71
4.3.2.5.3 State.....	71
4.3.2.6 Ausführen von vordefinierten Funktionen.....	72
4.3.2.6.1 BrbUaRsFireEvent	72
4.3.2.7 Zugriff auf die internen Daten	73
4.3.2.7.1 BrbUaRsGetNamespace.....	73
4.3.2.7.2 BrbUaRsGetEvent.....	73
4.3.2.7.3 BrbUaRsGetEventField	74
4.3.2.8 BrbUaRsMonitor.....	75
5 Anhänge	76
5.1 Hinweise zu dynamischen Arrays	76
5.2 Hinweise zu 64-Bit-Datentypen.....	76

1 Allgemeines

Die Bibliothek „BrbLibUa“ enthält nützliche Funktionen zum Betreiben einer OpcUa-Schnittstelle (sowohl Server als auch Client). Damit können Projekte übersichtlich und transparenter gestaltet werden.

Diese Bibliothek ist keine offizielle B&R-Software. Es besteht kein Anspruch auf Support, Wartung oder Fehlerbehebung. Die Benutzung geschieht auf eigene Gefahr.

Die Bibliothek unterliegt der MIT-Lizenz (siehe ‚License.txt‘), welche zwar unbeschränkte Nutzung auf eigene Gefahr gewährt, jedoch alle Haftungsansprüche ausschließt.

1.1 Hinweise zum Compiler

Das Entwicklungs- und Demo-Projekt ist auf den Compiler V6.3.0 gesetzt, mit dem das Projekt und damit auch die Bibliothek fehler- und warnungslos kompiliert werden können. Hinweis: Diese Version ist wegen der Implementierung der UnitTests nötig (siehe [Getestet mit UnitTests](#) unten).

Die Bibliothek ist aber auch unter älteren Compiler-Versionen einsetzbar.

1.2 Abhängigkeiten

Es besteht eine Abhängigkeit von folgenden Bibliotheken:

- BrbLib V5.03 oder höher
- AsOpcUac
- AsOpcUas
- DataObj
- FileIO

1.3 Hinweise zu StructuredText und anderen IEC-Sprachen

Die Bibliothek ist in ANSI-C geschrieben, kann aber auch in StructuredText und allen anderen IEC-Sprachen verwendet werden.

1.4 Getestet mit UnitTests

Das Automation Studio stellt ein Framework bereit, das die Möglichkeit bietet, Funktionen und Funktionsblöcke mit möglichst wenig Aufwand wiederholbar zu testen.

Dazu werden sogenannte (selbstprogrammierte) Unit-Tests verwendet. Ein Testfall ruft eine zu testende Funktion mit definierten Eingängen auf und vergleicht dann die Rückgabewert(e) mit dem zu erwartenden Ergebnis. Für eine Funktion können beliebig viele Testfälle implementiert werden. Vor allem werden damit Eingangs-Parameter im Grenzbereich des entsprechenden Datentyps getestet.

Aus allen Prüfungen wird automatisch ein Bericht erstellt, in welchem Fehlfunktionen schnell erfasst werden können.

So kann auch bei Neuerstellung/Änderung/Erweiterung einer Funktion die Fehlerfreiheit und die Kompatibilität schnell und sicher gewährleistet werden.

Für diese Bibliothek wurden für die meisten Funktionen und Funktionsblöcke viele Testfälle implementiert, was zur enormen Erhöhung der Software-Qualität führt.

1.5 Geprüft mit ClangTidy

Das gesamte Entwicklungs- und Demo-Projekt wurde mit dem Code-Analyse-Tool ClangTidy geprüft. Es erkennt ‚unschöne‘ Programm-Zeilen im AnsiC-Code, welche zwar nicht zu Compiler-Fehler oder -Warnungen führen, aber trotzdem nicht leicht erkennbare Fehlverhalten enthalten können. Zu diesem Zweck sind viele Regeln für die Code-Analyse definiert. Beispiele dafür sind:

Prüfung	Beispiel	Abhilfe
Verwendung von Literalen (Zahlenwerte)	Statische Zahlenwerte erschweren das Verständnis des Codes	Verwendung von Konstanten statt Literalen
Implizite Datentyp-Konvertierung	Zuweisung eines UDINT an eine	Einfügen expliziter, also ge-

(Casting)	UINT-Variable kann zu Datenverlust führen	wollter Konvertierungen oder Datentyp-Anpassung
Switch-Anweisung mit Enum	Ein Enum-Member wurde in einer Switch-Anweisung nicht berücksichtigt	Alle Member einer Enum in der Switch-Anweisung berücksichtigen
Vergleich von Datentypen mit und ohne Vorzeichen	Ein UDINT wird mit einem DINT verglichen	Sicherstellung des DINT-Werts > 0 oder Anpassung der Datentypen

ClangTidy analysiert den AnsiC -Code des gesamten Projekts und erstellt einen entsprechenden Bericht. Durch die Überprüfung bzw. Behebung der angezeigten Problemfälle wird die Software-Qualität der Bibliothek enorm erhöht.

Es gibt natürlich Code-Zeilen oder -Blöcke, welche zwar von ClangTidy erkannt werden, die aber als korrekt eingestuft werden können (z.B. weil es beim B&R-Compiler kein Problem verursacht oder weil es so gewollt ist). Damit diese nicht mehr im Bericht auftauchen, können durch Kommentare spezifizierte Analyse-Regeln umgangen werden. Beispiele sind:

```
// NOLINT (xxx)
// NOLINTNEXTLINE (xxx)
// NOLINTBEGIN (xxx)
// NOLINTEND (xxx)
```

Diese Kommentare dienen nur diesem Zweck und können vom Anwender einfach ignoriert werden.

1.6 Neueste Versionen auf GitHub

GitHub ist eine öffentliche Plattform für kostenlose Software. Der Download ist ohne Anmeldung möglich. Darauf sind verschiedene Pakete des Autors kostenlos erhältlich. Sie unterliegen alle der MIT-Lizenz (siehe oben).

Link zur neuesten Version des OpcUa-Sample-Projekts inklusive BrbLibUa:

<https://github.com/br-automation-com/OpcUaSamples-sample-AS/releases>

Die unterlagerte Bibliothek BrbLib ist als eigenes Release-Paket erhältlich. Es enthält neben dieser Bibliothek auch noch andere hilfreiche Bibliotheken in Sourcecode- und Binär-Version:

<https://github.com/br-automation-com/BrbLibs-lib-src/releases>

Auch erhältlich ist das Windows-Tool ‚RnCommTest‘ zum Testen von Kommunikationen. Es enthält u.a. folgende Module:

- Serielle Kommunikation (RS232/485)
- Tcp-Client, Tcp-Server
- Udp
- ModbusTcp-Master, ModbusTcp-Client
- OpcUa-Client**, OpcUa-Server, OpcUa-Subscriber, OpcUaUaBrMapper

Es ist unter diesem Link erhältlich:

<https://github.com/br-automation-com/RnCommTest-Windows/releases>

2 Revisionsgeschichte

2.1 BrbLibUa V1.03 – 2024-02-22

2.1.1 Hinweise zum Compiler

In diese Hilfe wurden die [Hinweise zum Compiler](#) aufgenommen.

2.1.2 Code-Prüfung mit ClangTidy

Die Prüfung mit ClangTidy des Entwicklungs- und Demo-Projekts (siehe Allgemeines/ [Geprüft mit ClangTidy](#)) wurde mit einer neuen Version durchgeführt (die vorige Version prüfte manche Tasks nicht komplett). Die dadurch erkannten Code-Stellen wurden überprüft und gegebenenfalls optimiert. Die Bibliothek wurde dabei nicht geändert.

2.1.3 Versionen geändert

Folgende Versionen wurden im Projekt geändert:

- AR von E4.91 auf H4.91

Auf die Funktion und die Kompatibilität hat dies keine Auswirkung.

2.1.4 Abhängigkeit geändert

Die Abhängigkeit von der Basis-Bibliothek „BrbLib“ wurde von V5.02 auf V5.03 geändert.

3 Fehlernummern

Einige der Fehlernummern, welche von Funktionen oder Funktionsblöcken im Fehlerfall zurückgegeben werden, sind in der Bibliothek BrbLib definiert und beschrieben.

Darüber hinaus sind zusätzlich in der Bibliothek BrbLibUa einige Fehlernummern definiert:

3.1 eBRB_ERR_UA_ERROR = 51000

Ein intern aufgerufener FB einer OpcUa-Bibliothek hat einen OpcUa-Fehler zurückgemeldet. Der zugehörige OpcUa-Status kann an einer anderen Stelle eingesehen werden.

Eine Liste der OpcUa-Stati und deren Codierung ist in der AS-Hilfe vorhanden (GUID=1e53f284-faa6-44bd-ac5e-7c88ec100ef8)

3.2 eBRB_ERR_UA_NO_ELEMENTS = 51001

Die Abfrage auf eine Auflistung enthält keine Elemente.

3.3 eBRB_ERR_UA_INVALID_INDEX = 51002

Bei der Abfrage auf eine Auflistung wurde ein ungültiger Index angegeben.

3.4 eBRB_ERR_UA_NOT_CONNECTED = 51003

Der Client hat keine Verbindung zum Server.

3.5 eBRB_ERR_UA_NOT_RUNNING = 51004

Der RunServer ist nicht im State eBRB_RSSTATE_RUNNING.

4 Pakete

4.1 General

In diesem Paket finden sich allgemeine Funktionen.

4.1.1 Status

4.1.1.1 BrbUaGetStatusCodeText

```
plcdword BrbUaGetStatusCodeText(unsigned long nStatusCode, plcstring* pStatusText, unsigned long nStatusTextSize)
```

Argumente:

UDINT nStatusCode
OpcUa-Status-Code
STRING* pStatusText
Zeiger auf den String, der gefüllt werden soll
UDINT nStatusTextSize
Größe des Strings, der gefüllt werden soll

Rückgabe:

DWORD
0x00000000 = Good (Kein Fehler)
0x803D0000 = Bad_NotSupported (Datentyp wird nicht unterstützt)
0x80460000 = Bad_StructureMissing (Nullpointer)

Beschreibung:

Die Funktion gibt den symbolischen Text eines OpcUa-Status-Codes zurück. Er ist sprechender als ein dezimaler Status-Code und daher zur Diagnose besser geeignet. In der AS-Hilfe sind die Codes als Hex beschrieben. Der Text wird in folgendem Format zurückgegeben:

„0xHex = Text“

Beispiele:

„0x00000000 = Good“
„0x80000000 = Bad“
„0x800A0000 = Bad_Timeout“
„0x80110000 = Bad_DataTypeIdUnknown“
„0x801F0000 = Bad_UserAccessDenied“

4.1.2 Nodes

In diesem Paket finden sich Datentypen und Funktionen für Knoten.

4.1.2.1 BrbUaSetNodeId

```
plcdword BrbUaSetNodeId(struct UANodeID* pNodeId, plcstring* sIdentifier, unsigned short nNamespaceIndex)
```

Argumente:

struct UANodeID* pNodeId
Zeiger auf eine NodeId-Struktur
STRING* sIdentifier
Zeiger auf den Identifier
UINT nNamespaceIndex
Namensraum-Index

Rückgabe:

DWORD
0x00000000 = Good (Kein Fehler)
0x80460000 = Bad_StructureMissing (Nullpointer)

Beschreibung:

Die Funktion besetzt eine NodeId-Struktur mit den übergebenen Werten. Enthält der Identifier eine Zahl, wird numerische Adressierung gesetzt, ansonsten textuelle.

4.1.2.2 BrbUaSetNodeIdNum

```
plcdword BrbUaSetNodeIdNum(struct UANodeID* pNodeId, unsigned long nIdentifier, unsigned short nNamespaceIndex)
```

Argumente:

```
struct UANodeID* pNodeId    Zeiger auf eine NodeId-Struktur
UDINT nIdentifier           Numerischer Identifier
UINT nNamespaceIndex        Namensraum-Index
```

Rückgabe:

```
DWORD
0x00000000 = Good           (Kein Fehler)
0x80460000 = Bad_StructureMissing (Nullpointer)
```

Beschreibung:

Die Funktion besetzt eine NodeId-Struktur mit den übergebenen Werten. Dabei wird immer die numerische Adressierung gesetzt.

4.1.2.3 BrbUaGetRandomNodeId

```
plcdword BrbUaGetRandomNodeId(struct UANodeID* pNodeId, enum UAIdentifierType eIdentifierType)
```

Argumente:

```
struct UANodeID* pNodeId    Zeiger auf eine NodeId-Struktur
enum UAIdentifierType        Identifier-Typ
```

Rückgabe:

```
DWORD
0x00000000 = Good           (Kein Fehler)
0x803D0000 = Bad_NotSupported (Identifier-Typ wird nicht unterstützt)
0x80460000 = Bad_StructureMissing (Nullpointer)
```

Beschreibung:

Die Funktion besetzt eine NodeId-Struktur mit zufälligen Werten. Sie kann zum Test verwendet werden.
Der IdentifierType `Opaque` wird nicht unterstützt.

4.1.2.4 BrbUaAreNodeIdsEqual

```
plcbits BrbUaAreNodeIdsEqual(struct UANodeID* pNodeId1, struct UANodeID* pNodeId2)
```

Argumente:

```
struct UANodeID* pNodeId1    Zeiger auf die 1. NodeId-Struktur
struct UANodeID* pNodeId2    Zeiger auf die 2. NodeId-Struktur
```

Rückgabe:

```
BOOL
0 = Ungleich
1 = Gleich
```

Beschreibung:

Die Funktion vergleicht 2 NodeId's auf Gleichheit. Bei Übergabe von Null-Pointern wird immer 0 zurückgegeben.

4.1.2.5 BrbUaAddNodeIdText

```
plcdword BrbUaAddNodeIdText(struct UANodeID* pNodeId, plcstring* pText, unsigned long nTextSize)
```

Argumente:

```
struct UANodeID* pNodeId  
    Zeiger auf eine NodeId-Struktur  
STRING* pText  
    Zeiger auf den String, der den Text aufnimmt  
UDINT nTextSize  
    Größe des Strings
```

Rückgabe:

```
DWORD  
0x00000000 = Good          (Kein Fehler)  
0x80460000 = Bad_StructureMissing (Nullpointer)
```

Beschreibung:

Die Funktion wandelt eine NodeId in einen lesbaren Text um und hängt diesen an den übergebenen String an.

Es wird das von der Foundation bevorzugte Format verwendet:

Namespace-Indizes werden immer mit `ns=` angegeben, auch der Namespace-Index 0.

Je nach Identifier-Typ folgt dann der Identifier:

```
Numeric      ; i=  
String       ; s=  
Guid         ; g=  
Opaque       ; o=
```

Beispiele:

```
ns=0;i=2252  
ns=6;s>::AxisX.rSpeed
```

So können NodeId's bequem visualisiert werden.

4.1.3 Attributes

In diesem Paket finden sich Datentypen und Funktionen für Attribute.

4.1.3.1 BrbUaGetAttributeIdDatatype

```
enum UAVariantType BrbUaGetAttributeIdDatatype(enum UAAttributeId eAttributeId)
```

Argumente:

```
enum UAAttributeId eAttributeId  
    Attribut
```

Rückgabe:

```
enum UAVariantType  
    Datentyp als Enum
```

Beschreibung:

Die Funktion gibt den OpcUa-Datentypen für ein Attribut zurück.

Da die möglichen Attribute von der Foundation vorgegeben sind, sind auch deren Datentypen statisch festgelegt.

Folgende Attribute sind möglich:

Attributes defined through the OPC-UA standard		
UAAI_Default	0	
UAAI_NodeId	1	
UAAI_NodeClass	2	
UAAI_BrowseName	3	
UAAI_DisplayName	4	
UAAI_Description	5	
UAAI_WriteMask	6	
UAAI_UserWriteMask	7	
UAAI_IsAbstract	8	
UAAI_Symmetric	9	
UAAI_InverseName	10	
UAAI_ContainsNoLoops	11	
UAAI_EventNotifier	12	
UAAI_Value	13	
UAAI_DataType	14	
UAAI_ValueRank	15	
UAAI_ArrayDimensions	16	
UAAI_AccessLevel	17	
UAAI_UserAccessLevel	18	
UAAI_MinimumSamplingInterval	19	
UAAI_Historizing	20	
UAAI_Executable	21	
UAAI_UserExecutable	22	

Folgende Rückgaben sind möglich:

Kind of the variant data		
UAVariantType_Null	0	No data
UAVariantType_Boolean	1	Data in element Boolean
UAVariantType_SByte	2	Data in element SByte
UAVariantType_Byte	3	Data in element Byte
UAVariantType_Int16	4	Data in element Int16
UAVariantType_UInt16	5	Data in element UInt16
UAVariantType_Int32	6	Data in element Int32
UAVariantType_UInt32	7	Data in element UInt32
UAVariantType_Int64	8	Data in element Int64
UAVariantType_UInt64	9	Data in element UInt64
UAVariantType_Float	10	Data in element Float
UAVariantType_Double	11	Data in element Double
UAVariantType_String	12	Data in element String
UAVariantType_DateTime	13	Data in element DateTime
UAVariantType_Guid	14	Guid data in element String
UAVariantType_ByteString	15	Byte string in element String
UAVariantType_XmlElement	16	XML data in element String
UAVariantType_NodeId	17	Data in element NodeId
UAVariantType_ExpandedNodeId	18	Data in element ExpandedNodeId
UAVariantType_StatusCode	19	Status code in element UInt32
UAVariantType_QualifiedName	20	Data in element QualifiedName
UAVariantType_LocalizedText	21	Data in element LocalizedText

4.1.3.2 BrbUaGetAttributeList

```
plcdword BrbUaGetAttributeList(struct UANodeInfo* pNodeInfo, struct BrbUANodeInfoAttributes_TYP* pAttributes)
```

Argumente:

```
struct UANodeInfo* pNodeInfo  
    Zeiger auf die NodeInfo  
struct BrbUANodeInfoAttributes_TYP* pAttributes  
    Zeiger auf die Liste der Attribute
```

Rückgabe:

DWORD

0x00000000 = Good (Kein Fehler)
0x80460000 = Bad_StructureMissing (Nullpointer)

Beschreibung:

Mit dem FB `UA_NodeGetInfo` der System-Bibliothek `AsOpcUac` können alle Attribute-Werte eines Knotens auf einmal ermittelt werden.

Um diese bequem in einer Visualisierung anzeigen zu können, kann mit dieser Funktion ein Array gefüllt werden, das pro Eintrag den Namen und den Wert-Text eines Attributes enthält.

Dazu wird der Zeiger auf die vorher ermittelte `NodeInfo` übergeben.

Außerdem muss der Anwender eine Instanz der folgenden Struktur anlegen und dessen Zeiger ebenfalls übergeben:

BrbUaNodeInfoAttributes_TYP	Attribute-Liste der Infos
Attribute	BrbUaNodeInfoAttribute_TYP[0..nBRBUA_ATTRIBUTES_INDEX_MAX]
	Attribute-Liste der Infos

Es enthält 22 Einträge mit diesen Angaben:

BrbUaNodeInfoAttribute_TYP	Attribut
sName	STRING[nBRBUA_ATTRIBUTE_NAME_CHAR_MAX]
sValue	STRING[nBRBUA_VALUE_TEXT_CHAR_MAX]
	Name des Attributs
	Wert des Attributs als Text

Die Werte der Attribute werden automatisch in Klartext gewandelt, so dass das Array bequem in einer Visualisierung angezeigt werden kann (z.B. zur Diagnose):

Attribute	BrbUaNodeInfoAttribute_TYP[0..22]	
Attribute[0]	BrbUaNodeInfoAttribute_TYP	
sName	STRING[32]	'Default'
sValue	STRING[255]	''
Attribute[1]	BrbUaNodeInfoAttribute_TYP	
sName	STRING[32]	'NodeId'
sValue	STRING[255]	'ns=8;s=:ServerData:anMultiDimArray'
Attribute[2]	BrbUaNodeInfoAttribute_TYP	
sName	STRING[32]	'NodeClass'
sValue	STRING[255]	'Variable'
Attribute[3]	BrbUaNodeInfoAttribute_TYP	
sName	STRING[32]	'BrowseName'
sValue	STRING[255]	'8.anMultiDimArray'
Attribute[4]	BrbUaNodeInfoAttribute_TYP	
sName	STRING[32]	'DisplayName'
sValue	STRING[255]	'anMultiDimArray'
Attribute[5]	BrbUaNodeInfoAttribute_TYP	
sName	STRING[32]	'Description'
sValue	STRING[255]	''
Attribute[6]	BrbUaNodeInfoAttribute_TYP	
sName	STRING[32]	'WriteMask'
sValue	STRING[255]	'0'
Attribute[7]	BrbUaNodeInfoAttribute_TYP	
sName	STRING[32]	'UserWriteMask'
sValue	STRING[255]	'0'
Attribute[8]	BrbUaNodeInfoAttribute_TYP	
sName	STRING[32]	'IsAbstract'
sValue	STRING[255]	'False'
Attribute[9]	BrbUaNodeInfoAttribute_TYP	
sName	STRING[32]	'Symmetric'
sValue	STRING[255]	'False'
Attribute[10]	BrbUaNodeInfoAttribute_TYP	
sName	STRING[32]	'InverseName'
sValue	STRING[255]	''
Attribute[11]	BrbUaNodeInfoAttribute_TYP	
sName	STRING[32]	'ContainsNoLoops'
sValue	STRING[255]	'False'
Attribute[12]	BrbUaNodeInfoAttribute_TYP	
sName	STRING[32]	'EventNotified'

Hinweis: Besitzt ein Knoten aufgrund seiner NodeClass ein Attribut nicht, so wird dessen Wert als leerer String angezeigt.

4.1.3.3 BrbUaAddBooleanText

```
plcdword BrbUaAddBooleanText(plcbit bBoolean, plcstring* pText, unsigned long nTextSize)
```

Argumente:

BOOL bBoolean
Boolescher Wert
STRING* pText
Zeiger auf den String, der den Text aufnimmt
UDINT nTextSize
Größe des Strings

Rückgabe:

DWORD
0x00000000 = Good (Kein Fehler)
0x80460000 = Bad_StructureMissing (Nullpointer)

Beschreibung:

Diese Funktion wandelt einen booleschen Wert in einen lesbaren Klartext um und hängt diesen an den übergebenen String an.

FALSE	=	„False“
TRUE	=	„True“

4.1.3.4 BrbUaAddNodeClassText

```
plcdword BrbUaAddNodeClassText(enum UANodeClass eNodeClass, plcstring* pText, unsigned long nTextSize)
```

Argumente:

enum UANodeClass eNodeClass
NodeClass
STRING* pText
Zeiger auf den String, der den Text aufnimmt
UDINT nTextSize
Größe des Strings

Rückgabe:

DWORD
0x00000000 = Good (Kein Fehler)
0x80460000 = Bad_StructureMissing (Nullpointer)

Beschreibung:

Der Wert des Attributs `NodeClass` wird als `Int32` ausgeliefert und entspricht der Enumeration `UANodeClass` aus der System-Bibliothek `AsOpcUac`.

Diese Funktion wandelt den Wert in einen lesbaren Klartext um und hängt diesen an den übergebenen String an, z.B.:

1	=	„Object“
4	=	„Method“
64	=	„DataType“

4.1.3.5 BrbUaAddDatatypeIdText

```
plcdword BrbUaAddDatatypeIdText(unsigned long nDatatypeId, plcstring* pText, unsigned long nTextSize)
```

Argumente:

UDINT nDatatypeId
Numerischer Identifier des Attributs 'Datatype'
STRING* pText
Zeiger auf den String, der den Text aufnimmt
UDINT nTextSize

Größe des Strings

Rückgabe:

DWORD

0x00000000 = Good (Kein Fehler)
0x80460000 = Bad_StructureMissing (Nullpointer)

Beschreibung:

Der Wert des Attributs `Datatype` wird als NodeId mit numerischer Id ausgeliefert (es zeigt auf den von der Foundation spezifizierten Typ-Knoten).

Diese Funktion wandelt den numerischen Identifier in einen lesbaren Klartext um und hängt diesen an den übergebenen String an, z.B.:

1	=	„1=Boolean“
4	=	„4=Int16“
21	=	„21=LocalizedText“

Bei benutzerdefinierten oder unbekannten Datentypen wird lediglich die Id in einen Text gewandelt.

4.1.3.6 BrbUaAddArrayDimensionText

```
plcdword BrbUaAddArrayDimensionText(unsigned long* pArrayDimension, signed long nValueRank, plcstring* pText, unsigned long nTextSize)
```

Argumente:

UDINT* pArrayDimension
Zeiger auf das UDINT-Array
DINT nValueRank
Anzahl der tatsächlichen Dimensionen
STRING* pText
Zeiger auf den String, der den Text aufnimmt
UDINT nTextSize
Größe des Strings

Rückgabe:

DWORD

0x00000000 = Good (Kein Fehler)
0x800F0000 = Bad_NothingToDo (nValueRank = 0)
0x80460000 = Bad_StructureMissing (Nullpointer)

Beschreibung:

Der Wert des Attributs `ArrayDimensions` wird als UDINT-Array ausgeliefert.

Diese Funktion wandelt dieses Array in einen lesbaren Klartext um und hängt diesen an den übergebenen String an. Der Parameter `nValueRank` wird intern auf 7 begrenzt.

Beispiel:

UDINT-Array:

		ArrayDimension	UDINT[0..6]	
	◆	ArrayDimension[0]	UDINT	3
	◆	ArrayDimension[1]	UDINT	5
	◆	ArrayDimension[2]	UDINT	7
	◆	ArrayDimension[3]	UDINT	0
	◆	ArrayDimension[4]	UDINT	0
	◆	ArrayDimension[5]	UDINT	0
	◆	ArrayDimension[6]	UDINT	0

nValueRank = 3

Ausgabe: „3;5;7“

4.1.3.7 BrbUaAddAccessLevelText

```
plcdword BrbUaAddAccessLevelText(plcbyte nAccessLevel, plcstring* pText, unsigned long nTextSize)
```

Argumente:

USINT nAccessLevel
AccessLevel

`STRING*` pText
Zeiger auf den String, der den Text aufnimmt
`UDINT` nTextSize
Größe des Strings

Rückgabe:

`DWORD`
0x00000000 = Good (Kein Fehler)
0x80460000 = Bad_StructureMissing (Nullpointer)

Beschreibung:

Der Wert des Attributs `AccessLevel` oder `UserAccessLevel` wird als bitcodiertes Byte ausgeliefert.

Diese Funktion wandelt den Wert in einen lesbaren Klartext um und hängt diesen an den übergebenen String an, z.B. wird der Byte-Wert 3 zu „3= Read, Write“.

4.1.3.8 BrbUaAddEventNotifierText

`plcdword` BrbUaAddEventNotifierText(`plcbyte` nEventNotifier, `plcstring*` pText, `unsigned long` nTextSize)

Argumente:

`USINT` nEventNotifier
Numerischer Identifier des Attributs 'Datatype'
`STRING*` pText
Zeiger auf den String, der den Text aufnimmt
`UDINT` nTextSize
Größe des Strings

Rückgabe:

`DWORD`
0x00000000 = Good (Kein Fehler)
0x80460000 = Bad_StructureMissing (Nullpointer)

Beschreibung:

Der Wert des Attributs `EventNotifier` wird als bitcodiertes Byte ausgeliefert.

Diese Funktion wandelt den Wert in einen lesbaren Klartext um und hängt diesen an den übergebenen String an, z.B. wird der Byte-Wert 3 zu „3= Subscribe, HistoryRead“.

4.1.4 VariantValues

In diesem Paket finden sich Datentypen und Funktionen für variante Werteverwaltung.

4.1.4.1 Datentyp

Der Wert eines Knotens kann sehr unterschiedliche Datentypen haben. Auf einer Sps kann daher eine Struktur verwendet werden, welche alle diese Standard-Datentypen abbilden kann. Diese ist in der Client-Bibliothek `AsOpcUac` definiert:

UAVariantData				Standard type for variant data
VariantType	UAVariantType	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Boolean	BOOL	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
SByte	SINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Byte	USINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Int16	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
UInt16	UINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Int32	DINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
UInt32	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Float	REAL	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Double	LREAL	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
DateTime	DATE_AND_TIME	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
String	STRING[255]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
NodeId	UANodeID	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
ExpandedNodeId	UAExpandedNodeID	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
QualifiedName	UAQualifiedName	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
LocalizedText	UALocalizedText	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
DataValue	UADataValue	<input type="checkbox"/>	<input checked="" type="checkbox"/>	

Der verwendete Datentyp ist als Enumeration im Element `VariantType` festgelegt. Die Enumeration lautet wie folgt:

1	2	3	4
UAVariantType			Kind of the variant data
UAVariantType_Null	0		No data
UAVariantType_Boolean	1		Data in element Boolean
UAVariantType_SByte	2		Data in element SByte
UAVariantType_Byte	3		Data in element Byte
UAVariantType_Int16	4		Data in element Int16
UAVariantType_UInt16	5		Data in element UInt16
UAVariantType_Int32	6		Data in element Int32
UAVariantType_UInt32	7		Data in element UInt32
UAVariantType_Int64	8		Data in element Int64
UAVariantType_UInt64	9		Data in element UInt64
UAVariantType_Float	10		Data in element Float
UAVariantType_Double	11		Data in element Double
UAVariantType_String	12		Data in element String
UAVariantType_DateTime	13		Data in element DateTime
UAVariantType_Guid	14		Guid data in element String
UAVariantType_ByteString	15		Byte string in element String
UAVariantType_XmlElement	16		XML data in element String
UAVariantType_NodeId	17		Data in element NodeId
UAVariantType_ExpandedNodeId	18		Data in element ExpandedNodeId
UAVariantType_StatusCode	19		Status code in element UInt32
UAVariantType_QualifiedName	20		Data in element QualifiedName
UAVariantType_LocalizedText	21		Data in element LocalizedText

Je nach Datentyp muss das entsprechende Element der Variant-Struktur benutzt werden (siehe dazu auch AS-Hilfe, GUID= 879518b2-71cd-4f0d-8d19-89766b5cf2a7).

4.1.4.2 BrbUaClearVariantValue

```
plcdword BrbUaClearVariantValue(struct UAVariantData* pUaVariantData)
```

Argumente:

```
struct UAVariantData* pUaVariantData
    Zeiger auf eine Variant-Struktur
```

Rückgabe:

```
DWORD
0x00000000 = Good                (Kein Fehler)
0x80460000 = Bad_StructureMissing (Nullpointer)
```

Beschreibung:

Die Funktion setzt den Inhalt der gesamten Struktur auf „0“, belässt aber die Angabe des Datentyps `VariantType`.

4.1.4.3 BrbUaConvVariantValueFromString

```
plcdword BrbUaConvVariantValueFromString(struct UAVariantData* pUaVariantData, plcstring* pValueText)
```

Argumente:

```
struct UAVariantData* pUaVariantData
    Zeiger auf eine Variant-Struktur
STRING* pValueText
    Zeiger auf den String, der den Wert enthält
```

Rückgabe:

```
DWORD
0x00000000 = Good                (Kein Fehler)
0x80330000 = Bad_NodeIdInvalid   (Syntaxfehler bei NodeId)
0x803D0000 = Bad_NotSupported    (Datentyp wird nicht unterstützt)
0x80460000 = Bad_StructureMissing (Nullpointer)
```

Beschreibung:

Die Funktion wandelt einen Text in einen Variant-Wert. Der Datentyp muss in der Struktur angegeben sein.

Achtung: Es werden nicht alle Datentypen unterstützt. Hier die Auflistung der Datentypen:

Variant-Datentyp	Variant-Struktur-Element	Bemerkung
Null	-	Liefert Bad_NotSupported
Boolean	Boolean	
SByte	SByte	
Byte	Byte	
Int16	Int16	
UInt16	UInt16	
Int32	Int32	
UInt32	UInt32	
Int64	-	Liefert Bad_NotSupported
UInt64	-	Liefert Bad_NotSupported
Float	Float	
Double	Double	Wird erst in Float und dann in Double gewandelt
String	String	
DateTime	DateTime	Benötigtes Format steht in Konstante sBRB_DATETIME_FORMAT
Guid	String	
ByteString	-	Liefert Bad_NotSupported
XmlElement	-	Liefert Bad_NotSupported
NodeId	NodeId	Möglich sind hier die Identifier-Typen Numeric, String und Guid. Opaque wird nicht unterstützt.
ExpandedNodeId	-	Liefert Bad_NotSupported
StatusCode	UInt32	
QualifiedName		
LocalizedText		

4.1.4.4 BrbUaConvVariantValueToString

```
plcdword BrbUaConvVariantValueToString (struct UAVariantData* pUaVariantData, plcstring* pValueText, unsigned long nValueTextSize)
```

Argumente:

```
struct UAVariantData* pUaVariantData  
    Zeiger auf eine Variant-Struktur  
STRING* pValueText  
    Zeiger auf den String, der gefüllt werden soll  
UDINT nValueTextSize  
    Größe des Strings, der gefüllt werden soll
```

Rückgabe:

```
DWORD  
0x00000000 = Good (Kein Fehler)  
0x803D0000 = Bad_NotSupported (Datentyp wird nicht unterstützt)  
0x80460000 = Bad_StructureMissing (Nullpointer)
```

Beschreibung:

Die Funktion wandelt einen Variant-Wert in einen Text. Der Datentyp muss in der Struktur angegeben sein.

Achtung: Es werden nicht alle Datentypen unterstützt. Hier die Auflistung der Datentypen:

Variant-Datentyp	Variant-Struktur-Element	Bemerkung
Null	-	Liefert „Null“
Boolean	Boolean	Liefert „True“ oder „False“
SByte	SByte	
Byte	Byte	
Int16	Int16	
UInt16	UInt16	
Int32	Int32	
UInt32	UInt32	
Int64	-	Liefert Bad_NotSupported
UInt64	-	Liefert Bad_NotSupported
Float	Float	

Double	Double	Wird erst in Float und dann in Text gewandelt
String	String	
DateTime	DateTime	Benötigtes Format steht in Konstante sBRB_DATETIME_FORMAT
Guid	String	
ByteString	-	Liefert Bad_NotSupported
XmlElement	-	Liefert Bad_NotSupported
NodeId	NodeId	Möglich sind hier die Identifier-Typen Numeric, String und Guid. Opaque wird nicht unterstützt.
ExpandedNodeId	-	Liefert Bad_NotSupported
StatusCode	UInt32	
QualifiedName	-	Liefert Bad_NotSupported
LocalizedText	-	Liefert Bad_NotSupported

4.1.4.5 BrbUaAddVariantValueSubName

```
plcdword BrbUaAddVariantValueSubName(enum UAVariantType eVariantType, plcstring* pText, unsigned long nTextSize)
```

Argumente:

enum UAVariantType eVariantType
Angabe des Datentyps
STRING* pText
Zeiger auf den String, an den angehängt werden soll
UDINT nTextSize
Größe des Strings

Rückgabe:

DWORD
0x00000000 = Good (Kein Fehler)
0x803D0000 = Bad_NotSupported (Datentyp wird nicht unterstützt)
0x80460000 = Bad_StructureMissing (Nullpointer)

Beschreibung:

Manchmal muss der Name einer Variablen an einem Funktionsblock der OpcUa-Bibliotheken angegeben werden. Wenn es sich dabei um einen Variant handelt, muss auf das Element des verwendeten Datentyps verwiesen werden.

Diese Funktion hängt an einen String den Elementnamen des Variants aufgrund seines Datentyps an. Beispiel:

```
UAVariantType_Boolean -> „.Boolean“  
UAVariantType_Float -> „.Float“
```

4.1.5 ByteStrings

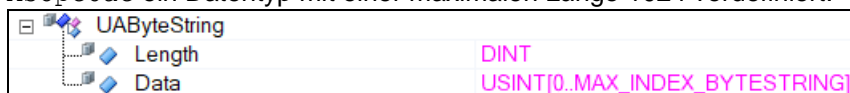
In diesem Paket finden sich Datentypen und Funktionen zur Behandlung von ByteStrings.

4.1.5.1 Grundsätzliches zum Datentyp ByteString

Der OpcUa-Datentyp `ByteString` wird verwendet, um ein dynamisches (also in der Länge veränderbares) Byte-Array abzubilden. Siehe dazu [Hinweise zu dynamischen Arrays](#).

Auf einer SPS muss ein Array aber statisch mit fester Länge zur Entwurfszeit festgelegt werden.

Um auf einer SPS dennoch mit dynamischen Arrays umgehen zu können, ist in der System-Bibliothek `AsOpcUac` ein Datentyp mit einer maximalen Länge 1024 vordefiniert:



Die tatsächlich gültige Länge muss dabei in `Length` stehen. Damit können also Byte-Arrays mit dynamischer Länge (0..1024) behandelt werden.

Sollten weniger oder mehr Elemente benötigt werden, kann dieser Datentyp auch abgeleitet werden.

Da bei den folgenden Funktionen immer beide Elemente dieser Struktur übergeben werden, können damit auch die abgeleiteten Datentypen behandelt werden (z.B. eine `EventId`, siehe unten).

4.1.5.2 Datentyp UABigIntegerBrbEventId_TYP

Eine EventId ist eine eindeutige, sehr große Identifizierungs-Nummer für ein Event. Man benötigt 16 Bytes, um diese Zahl zu speichern. Sie hat also einen Bereich von 0..~3e+38 (zum Vergleich: Ein UDINT mit 4 Bytes hat einen Bereich von 0..4294967295).

Auf einer SPS wird eine EventId in einem ByteString mit Länge 16 gespeichert. Tatsächlich werden EventId's bei den Funktionen der System-Bibliothek AsOpcUac so übergeben.

Da der Datentyp UABigInteger aus der System-Bibliothek AsOpcUac (siehe oben) 1024 Bytes hat und damit unverhältnismäßig viel Speicher benötigen würde, wurde in der BrbLibUa ein Datentyp mit der Länge 16 abgeleitet:

UABigIntegerBrbEventId_TYP		EventId als ByteString mit Länge 16
Length	DINT	Länge der Daten (sollte mit 16 besetzt werden)
Data	USINT[0..nBRBUA_EVENTID_INDEX_MAX]	Bytes der EventId
nBRBUA_EVENTID_INDEX_MAX	UINT	15
		Maximaler Index der Bytes einer EventId

Damit das Betriebssystem den Datentyp als Ableitung erkennt, konnte nicht das bibliothekstypische Präfix „Brb“ verwendet werden.

Hinweis: Jedes vom Server gefeuerte Event muss eine eindeutige EventId haben. Jedes SDK, so auch die B&R-SPS, bedient sich dabei eines mathematischen Tricks aus der Statistik: Um eine einmalig eindeutige ID zu erhalten, werden alle Daten-Bytes mit Zufallszahlen besetzt. Aufgrund der Größe der Id ist die Wahrscheinlichkeit, dass dabei im Lebens-Zyklus einer SPS zweimal dieselbe Id generiert wird, sehr, sehr gering. Zur applikativen Generierung kann die Funktion BrbGetRandomByteString (siehe unten) verwendet werden.

4.1.5.3 Umwandlung eines ByteStrings in Hex-Darstellung

Um einen ByteString (z.B. eine EventId) bequem anzuzeigen, bietet es sich an, diesen in einen Hex-String zu wandeln (die meisten OpcUa-Tools zeigen eine EventId als Hex-String an). Manchmal kann es auch nötig sein, einen Hex-String in einen ByteString zu wandeln.

Dafür sind Funktionen in der Basis-Bibliothek BrbLib enthalten: BrbUsintArrayToHex() und BrbHexToUsintArray().

4.1.5.4 BrbUaIncByteString

```
plcdword BrbUaIncByteString(unsigned char* pData, signed long nLength)
```

Argumente:

USINT* pData
Zeiger auf das Daten-Array des ByteStrings
DINT nLength
Länge des Daten-Arrays

Rückgabe:

DWORD
0x00000000 = Good (Kein Fehler)
0xB00C0000 = PlcOpen_BadElementCount (Überlauf, Beginn wieder bei 0)
0x803C0000 = Bad_OutOfRange (Length = 0)
0x80460000 = Bad_StructureMissing (Nullpointer)

Beschreibung:

Ein ByteString kann als Zahl interpretiert werden (z.B. als EventId, siehe oben).

Diese Funktion inkrementiert den ByteString um 1.

Dabei wird das letzte Byte um 1 erhöht. Steht dieses schon auf 255, so wird es auf 0 gesetzt und das nächste Byte um 1 erhöht. Dieser Überlauf wird bei jedem Byte angewendet. Sind alle Bytes auf 255, werden alle Bytes wieder auf 0 gesetzt. Die Angabe nLength wird dabei beachtet.

4.1.5.5 BrbUaGetRandomByteString

```
plcdword BrbUaGetRandomByteString(unsigned char* pData, signed long nLength)
```

Argumente:

USINT* pData
Zeiger auf das Daten-Array des ByteStrings
DINT nLength
Länge des Daten-Arrays

Rückgabe:

DWORD
0x00000000 = Good (Kein Fehler)
0x803C0000 = Bad_OutOfRange (Length = 0)
0x80460000 = Bad_StructureMissing (Nullpointer)

Beschreibung:

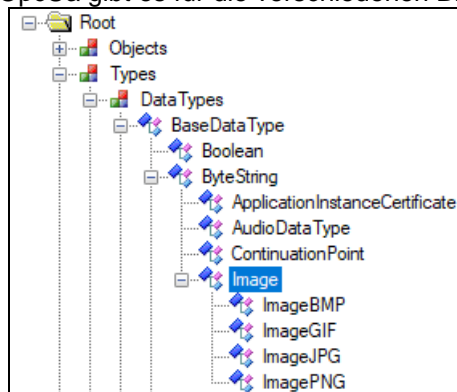
Erzeugt einen zufälligen ByteString mit angegebener Länge. Sie kann zum Test verwendet werden. Das Daten-Array muss natürlich mindestens diese Länge haben.

4.1.6 Images

In diesem Paket finden sich Datentypen und Funktionen für Bild-Formate.

4.1.6.1 Datentypen

Bei OpcUa gibt es für die verschiedenen Bild-Formate eigens spezifizierte Datentypen:



Diese sind vom Datentyp `ByteString` abgeleitet, welcher im Grunde nur ein dynamisches Byte-Array darstellt.

Bei B&R werden Bilder deshalb als dynamische USINT-Arrays verwaltet (siehe AS-Hilfe, GUID=da671bed-a169-43fb-8434-1f6c003c21de).

In der Client-Bibliothek `AsOpcUac` sind dafür schon Datentypen mit festen Längen definiert:

- BrUaImageBMP
- BrUaImageGIF
- BrUaImageJPG
- BrUaImagePNG

Bezeichnung	Datentyp	Beschreibung
Length	DINT	Länge der Bilddaten.
Data	ARRAY[0..6291455] OF USINT	<p>Sequenz von Byte-Werten welche ein Bild im BMP-Format beinhalten. Das OPC-UA System führt keine Validierung der Daten durch. Für das Format ungültige Bilddaten werden ohne Fehlermeldung durchgereicht.</p> <p>Die Größe von 6MB entspricht einer "Standard" BMP-Datei im Full HD Format (1920x1080). Entspricht diese Größe nicht dem gegebenen Anwendungsfall, ist eine anwendungsgerechte Abwandlung des Datentyps möglich.</p>

Es können aber auch eigene Datentypen mit einer bestimmten Größe definiert werden (siehe ebenfalls AS-Hilfe, GUID= da671bed-a169-43fb-8434-1f6c003c21de)

4.1.6.2 BrbUaLoadImage

```
void BrbUaLoadImage(struct BrbUaLoadImage* inst)
```

Argumente:

```
struct BrbUaLoadImage* inst  
    Zeiger auf die Funktionsblock-Instanz
```

Eingänge:

```
STRING* pDevice  
    Zeiger auf den Laufwerks-Namen  
STRING* pFile  
    Zeiger auf den Datei-Namen inkl. Pfad  
USINT* pImageData  
    Zeiger auf das Element ‚Data‘ der Bild-Struktur (Array der Bild-Daten)  
UDINT nImageDataSize  
    Größe des angegebenen Arrays (sizeof)  
DINT* pImageLength  
    Zeiger auf das Element ‚Length‘ der Bild-Struktur (Anzahl der Bytes)
```

Ausgänge:

```
UINT nStatus  
    Funktionsblock-Status  
eBRB_OK  
eBRB_ERR_NULL_POINTER  
eBRB_ERR_BUSY  
1..65534 = Fehlermeldung. Wenn es sich um einen System-Fehler (<50000) handelt, kann die Beschreibung dazu in der Studio-Hilfe gefunden werden.
```

Beschreibung:

Lädt die angegebene Bild-Datei in ein dynamisches USINT-Array. Natürlich muss das Array groß genug sein, um die Datei zu laden.

4.1.7 LocalizedTexts

In diesem Paket finden sich Datentypen und Funktionen für den OpcUa-Datentyp LocalizedText.

4.1.7.1 BrbUaSetLocalizedText

```
plcdword BrbUaSetLocalizedText(BrUaLocalizedText* pLocalizedText, plcstring* pLocale, plcstring* pText)
```

Argumente:

```
struct BrUaLocalizedText* pLocalizedText  
    Zeiger auf eine LocalizedText-Struktur  
STRING* pLocale  
    Zeiger auf den String, der die ‚Locale‘-Angabe enthält  
STRING* pText
```

Zeiger auf den String, der die ‚Text‘-Angabe enthält

Rückgabe:

DWORD

0x00000000 = Good (Kein Fehler)
0x80460000 = Bad_StructureMissing (Nullpointer)

Beschreibung:

Die Funktion besetzt den Inhalt der Struktur mit den angegebenen Werten. Der Code wird dadurch kürzer und transparenter.

4.1.7.2 BrbUaGetRandomLocalizedText

```
plcdword BrbUaGetRandomLocalizedText(BrUaLocalizedText* pLocalizedText, unsigned short nLength)
```

Argumente:

struct BrUaLocalizedText* pLocalizedText
Zeiger auf eine LocalizedText-Struktur
USINT nLength
Zu erzeugende Länge des Textes

Rückgabe:

DWORD

0x00000000 = Good (Kein Fehler)
0x80460000 = Bad_StructureMissing (Nullpointer)

Beschreibung:

Die Funktion gibt einen zufälligen LocalizedText zurück. Sie kann zum Test verwendet werden. Die Sprache wird dabei zufällig aus de, en, fr und es gewählt. Der Text enthält eine zufällige Kombination aus Zahlen, Groß- und Kleinbuchstaben mit der übergebenen Länge.

4.1.7.3 BrbUaAddLocalizedTextText

```
plcdword BrbUaAddLocalizedTextText(BrUaLocalizedText* pLocalizedText, plcstring* pText, unsigned long nTextSize)
```

Argumente:

struct BrUaLocalizedText* pLocalizedText
Zeiger auf eine LocalizedText-Struktur
STRING* pText
Zeiger auf den String, der den Text aufnimmt
UDINT nTextSize
Größe des Strings

Rückgabe:

DWORD

0x00000000 = Good (Kein Fehler)
0x80460000 = Bad_StructureMissing (Nullpointer)

Beschreibung:

Die Funktion wandelt einen LocalizedText in einen Klartext um und hängt diesen an den übergebenen String an.

Es wird das von der Foundation bevorzugte Format verwendet:

Locale:Text

Die beiden Teile werden also durch : getrennt. Wird kein Locale angegeben, entfällt auch das Trennzeichen.

Beispiele:

en:Hello
de:Hallo
es:Hola
Hello

So können LocalizedTexts bequem visualisiert werden.

4.1.8 QualifiedNames

In diesem Paket finden sich Datentypen und Funktionen für den OpcUa-Datentyp `QualifiedName`.

4.1.8.1 BrbUaSetQualifiedName

```
plcdword BrbUaSetQualifiedName (BrUaQualifiedName* pQualifiedName, unsigned short nNamespaceIndex,
plcstring* pName)
```

Argumente:

```
struct BrUaQualifiedName* pQualifiedName
    Zeiger auf eine QualifiedName-Struktur
UINT nNamespaceIndex
    Namensraum-Index
STRING* pName
    Zeiger auf den String, der die ,Name'-Angabe enthält
```

Rückgabe:

```
DWORD
    0x00000000 = Good                (Kein Fehler)
    0x80460000 = Bad_StructureMissing (Nullpointer)
```

Beschreibung:

Die Funktion besetzt den Inhalt der Struktur mit den angegebenen Werten. Der Code wird dadurch kürzer und transparenter.

4.1.8.2 BrbUaGetRandomQualifiedName

```
plcdword BrbUaGetRandomQualifiedName (BrUaQualifiedName* pQualifiedName, unsigned short nLength)
```

Argumente:

```
struct BrUaQualifiedName * pQualifiedName
    Zeiger auf eine QualifiedName-Struktur
USINT nLength
    Zu erzeugende Länge des Namens
```

Rückgabe:

```
DWORD
    0x00000000 = Good                (Kein Fehler)
    0x80460000 = Bad_StructureMissing (Nullpointer)
```

Beschreibung:

Die Funktion gibt einen zufälligen `QualifiedName` zurück. Sie kann zum Test verwendet werden. Der Namespace-Index wird dabei zufällig zwischen 0 und 65535 gewählt. Der Name enthält eine zufällige Kombination aus Zahlen, Groß- und Kleinbuchstaben mit der übergebenen Länge.

4.1.9 TimeAndDate

In diesem Paket finden sich Funktionen für Zeit und Zeitzonen, z.B. den OpcUa-Datentyp `UaSrv_DateTimeType`.

4.1.9.1 BrbDtStructToUaSrvDateTime

```
unsigned short BrbUaDtStructToUaSrvDateTime (struct UaSrv_DateTimeType* pUaSrvDateTime, struct
DTStructure* pDtStruct, unsigned short nNanoSeconds)
```

Argumente:

```
struct UaSrv_DateTimeType* pUaSrvDateTime
    Zeiger auf eine Instanz von "UaSrv_DateTimeType"
struct DTStructure* pDtStruct
    Zeiger auf eine Instanz von "DTStructure"
UINT nNanoSeconds
    Anzahl der Nano-Sekunden 0..999 (DTStructure hat nur eine Auflösung von Micro-Sekunden,
    UaSrv_DateTimeType aber von Nano-Sekunden).
```


Rückgabe:

```
UINT
    eBRB_ERR_OK = 0
    eBRB_ERR_NULL_POINTER = 50000
```

Beschreibung:

Wandelt eine Zeit im DTStructure-Format in ein UaSrv_DateTimeType-Format.

4.1.9.2 BrbUaSrvDateTimeToDtStruct

```
unsigned short BrbUaSrvDateTimeToDtStruct(struct DTStructure* pDtStruct, struct
    UaSrv_DateTimeType* pUaSrvDateTime)
```

Argumente:

```
struct DTStructure* pDtStruct
    Zeiger auf eine Instanz von "DTStructure"
struct UaSrv_DateTimeType* pUaSrvDateTime
    Zeiger auf eine Instanz von "UaSrv_DateTimeType"
```

Rückgabe:

```
UINT
    eBRB_ERR_OK = 0
    eBRB_ERR_NULL_POINTER = 50000
```

Beschreibung:

Wandelt eine Zeit im UaSrv_DateTimeType-Format in ein DTStructure-Format. Die Nano-Sekunden gehen dabei verloren.

4.1.9.3 BrbUaGetSrvTimeText

```
unsigned short BrbUaGetSrvTimeText(struct UaSrv_DateTimeType* pSrvTime, plcstring* pText, un-
    signed long nTextSize, signed short nTimeOffset, plcstring* pFormat)
```

Argumente:

```
struct DateTimeType* pSrvTime
    Zeiger auf eine Zeitangabe
STRING* pText
    Zeiger auf den String, der gefüllt werden soll
UDINT nTextSize
    Größe des Strings, der gefüllt werden soll
INT nTimeOffset
    Zeit-Offset zum Umrechnen von Utc- auf lokale Zeit in Minuten. Wenn Utc-Zeit gewünscht ist, dann 0
STRING* pFormat
    Zeiger auf den String, der die Formatierung enthält
        Jahr          „yyyy“ oder „yy“
        Monat         „mm“ oder „m“
        Tag           „dd“ oder „d“
        Stunde        „hh“ oder „h“
        Minute        „MM“ oder „M“
        Sekunde       „ss“ oder „s“
        Millisekunde  „mil“
        Mikrosekunde  „mic“
        Nanosekunde   „nan“
```

Rückgabe:

```
UINT
    eBRB_ERR_OK = 0
    eBRB_ERR_NULL_POINTER = 50000
    eBRB_ERR_INVALID_PARAMETER = 50001
```

Beschreibung:

Füllt einen String mit dem übergebenen Zeitstempel. Die oben genannten Schlüsselzeichen im Format-Text werden mit dem jeweiligen Wert ersetzt, andere Zeichen bleiben bestehen.

Hinweis: Ein OpcUa-Zeitstempel ermöglicht eine Auflösung auf 100ns. Die letzten beiden Stellen gehen bei der Übertragung verloren und werden daher immer mit 0 gefüllt.

4.1.9.4 BrbUaGetRandomTimezone

```
plcdword BrbUaGetRandomTimezone (BrbUaTimezoneDataType* pTimezone)
```

Argumente:

BrbUaTimezoneDataType* pTimezone
Zeiger auf die Zeit-Zone

Rückgabe:

DWORD

0x00000000 = Good (Kein Fehler)
0x80460000 = Bad_StructureMissing (Nullpointer)

Beschreibung:

Erzeugt eine zufällige Zeit-Zone. Sie kann zum Test verwendet werden.

4.1.10 ServerInfo

In diesem Paket befinden sich Struktur-Definitionen zum Lesen oder Abonnieren von Info-Daten. Damit kann z.B. die Software-Version des Servers ermittelt werden.

Die [Hinweise zu den Diagnose-Daten](#) gelten teilweise auch für die Info-Daten und sollten daher berücksichtigt werden.

Der applikative Zugriff auf den Datenpunkt ‚ServerState‘ (siehe unten) ist über die System-Funktion ‚AsOpcUas.UaSrv_GetServerState()‘ möglich.

Der Zugriff auf die kompletten Daten ist nur über eine Client-Verbindung möglich (es gibt keine System-Funktion dafür). Es kann aber ein Client implementiert werden, welcher sich auf den eigenen Server verbindet und so die Daten auslesen kann.

4.1.10.1 Namespace-Array

Beim Lesen/Abonnieren des Knotens
/Root/Objects/Server/NamespaceArray
= “ns=0;i=2255”

werden Daten gesendet, die in einer Instanz dieser Struktur abgebildet werden können:

BrbUaServerNamespacesArray_TYP					
Length	UAArrLength	<input type="checkbox"/>	<input checked="" type="checkbox"/>		Array für Namespaces
Data	STRING[nBRBUA_NAMESPACE_URI_CHAR_MAX][0..nBRBUA_SRVDIAG_NAMESPACE_IDX_MAX]	<input type="checkbox"/>	<input checked="" type="checkbox"/>		

Es handelt sich dabei um ein dynamisches Array (siehe auch [Hinweise zu dynamischen Arrays](#)), welches maximal 20 Elemente aufnehmen kann.

Achtung: Sind am Server mehr als 20 Namespaces bekannt, kann das empfangene Array nicht auf die Variable kopiert werden. Die Variable wird dann nicht mehr aktualisiert!

Müssen mehr Namespaces abgebildet werden können, muss ein eigener Datentyp mit einem größeren Array definiert werden.

Achtung: Hier genügt meistens ein einzelner Read nach dem Verbindungs-(Wieder)-Aufbau, da sich diese Liste nicht online, sondern nur (wenn überhaupt) nach Neustart des Servers ändert.

4.1.10.2 ServerStatus

Beim Lesen/Abonnieren des Knotens
/Root/Objects/Server/ServerStatus
= “ns=0;i=2256”

werden Daten gesendet, die in einer Instanz dieser Struktur abgebildet werden können:

BrbUaServerStatus_TYP					Status-Daten des Servers
dtStartTime	DATE_AND_TIME	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
dtCurrentTime	DATE_AND_TIME	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
eServerState	BrbUaServerState_ENUM	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
BuildInfo	BrbUaServerBuildInfo_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
nSecondsTillShutdown	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
ShutdownReason	UALocalizedText	<input type="checkbox"/>	<input checked="" type="checkbox"/>		

Die „BuildInfo“ befindet sich in einer eigenen Unterstruktur:

BrbUaServerBuildInfo_TYP			<input checked="" type="checkbox"/>		Build-Daten des Servers
sProductUri	STRING[255]	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
sManufacturerName	STRING[255]	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
sProductName	STRING[255]	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
sSoftwareVersion	STRING[255]	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
sBuildNumber	STRING[255]	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
dtBuildDate	DATE_AND_TIME	<input type="checkbox"/>	<input checked="" type="checkbox"/>		

Der „eServerState“ ist als Enumeration aufgelegt:

BrbUaServerState_ENUM
eBRBUA_SRVSTATE_RUNNING
eBRBUA_SRVSTATE_FAILED
eBRBUA_SRVSTATE_NOCONFIG
eBRBUA_SRVSTATE_SUSPENDED
eBRBUA_SRVSTATE_SHUTDOWN
eBRBUA_SRVSTATE_TEST
eBRBUA_SRVSTATE_COMMFAULT
eBRBUA_SRVSTATE_UNKNOWN

Genauere Beschreibung zu den Datenpunkten kann in der OpcUa-Spezifikation Part 5 nachgeschlagen werden.

Achtung: Nur einige Datenpunkte ändern sich zyklisch (z.B. ‚dtCurrentTime‘). Werden nur die statischen Datenpunkte benötigt (z.B. ‚sSoftwareVersion‘), genügt meistens ein einzelner Read nach dem Verbindungs-(Wieder)-Aufbau, da sich diese Werte nicht online, sondern nur (wenn überhaupt) nach Neustart des Servers ändern.

Wird z.B. nur ‚dtCurrentTime‘ benötigt, ist es performanter, nur diesen Datenpunkt und nicht die ganze Struktur zu abonnieren.

4.1.11 ServerDiag

In diesem Paket befinden sich Struktur-Definitionen zum Lesen oder Abonnieren von Diagnose-Daten. Damit kann z.B. ermittelt werden, wie viele und welche Clients am Server verbunden sind (siehe dazu unbedingt die [Hinweise zu den Diagnose-Daten](#)).

Der Zugriff auf die Daten ist nur über eine Client-Verbindung möglich (es gibt keine System-Bibliothek dafür). Es kann aber ein Client implementiert werden, welcher sich auf den eigenen Server verbindet und so die Daten auslesen kann.

Achtung: Prinzipiell werden Diagnose-Daten nur dann vom Server aktualisiert, wenn der Server dies unterstützt. Dies ist am booleschen Knoten

/Root/Objects/Server/ServerDiagnostics/EnabledFlag
= „ns=0;i=2294“

sichtbar. Ist dieser Knoten „False“ oder nicht vorhanden, werden die Diagnose-Daten nicht unterstützt. Bei einer B&R-SPS ist er „True“.

4.1.11.1 ServerDiagData

Beim Lesen/Abonnieren des Knotens

Root/Objects/Server/ServerDiagnostics/ServerDiagnosticsSummary
= „ns=0;i=2275“

werden Daten gesendet, die in einer Instanz dieser Struktur abgebildet werden können:

BrbUaServerDiagData_TYP			<input checked="" type="checkbox"/>	Diagnose-Daten einer Session
nServerViewCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nCurrentSessionCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nCumulatedSessionCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nSecurityRejectedSessionCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nRejectedSessionCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nSessionTimeoutCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nSessionAbortCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nCurrentSubscriptionCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nCumulatedSubscriptionCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nPublishingIntervalCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nSecurityRejectedRequestsCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nRejectedRequestsCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	

Hinweis: Die mit „Current“ gekennzeichneten Zähler geben jeweils den aktuellen Stand an, wogegen die mit „Cumulated“ gekennzeichneten nur hochzählen.

Genauere Beschreibung zu den Datenpunkten kann in der OpcUa-Spezifikation Part 5 nachgeschlagen werden.

4.1.11.2 SessionDiagData

Beim Lesen/Abonnieren des Knotens

Root/Objects/Server/ServerDiagnostics/SessionsDiagnosticsSummary/SessionDiagnosticsArray
= “ns=0;i=3707”

werden Daten gesendet, die in einer Instanz dieser Struktur abgebildet werden können:

BrbUaSessDiagArray_TYP			<input checked="" type="checkbox"/>	Array für Session-Diagnose
Length	UAArrayLength	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Data	BrbUaSessDiagData_TYP[0..nBRBUA_SRVDIAG_SESSION_INDEX_MAX]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	

Es handelt sich dabei um ein dynamisches Array (siehe auch [Hinweise zu dynamischen Arrays](#)), welches maximal 10 Elemente aufnehmen kann.

Achtung: Sind am Server mehr als 10 Sessions aktiv, kann das empfangene Array nicht auf die Variable kopiert werden. Die Variable wird dann nicht mehr aktualisiert!

Müssen mehr Sessions abgebildet werden können, muss ein eigener Datentyp mit einem größeren Array definiert werden.

Ein Element beschreibt eine Session, also eine Client-Verbindung:

BrbUaSessDiagData_TYP			<input checked="" type="checkbox"/>	Diagnose-Daten einer Session
SessionId	UAnodeId	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
sSessionName	STRING[255]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
ClientDescription	BrbUaApplicationDescription_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
sServerUri	STRING[255]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
sEndpointUri	STRING[255]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
LocaleIdsCount	UAnoOfElements	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
sLocaleIds	STRING[7][0..4]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
rActualSessionTimeout	LREAL	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nMaxResponseMessageSize	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
dtClientConnectionTime	DATE_AND_TIME	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
dtClientLastContactTime	DATE_AND_TIME	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nCurrentSubscriptionsCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nCurrentMonitoredItemsCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nCurrentPublishRequestsInQueue	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
TotalRequestCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nUnauthorizedRequestCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
ReadCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
HistoryReadCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
WriteCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
HistoryUpdateCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
CallCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
CreateMonitoredItemsCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
ModifyMonitoredItemsCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
SetMonitoringModeCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
SetTriggeringCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
DeleteMonitoredItemsCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
CreateSubscriptionCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
ModifySubscriptionCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
SetPublishingModeCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
PublishCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
RepublishCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
TransferSubscriptionsCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
DeleteSubscriptionsCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
AddNodesCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
AddReferencesCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
DeleteNodesCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
DeleteReferencesCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
BrowseCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
BrowseNextCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
TranslateBrowsePathToNodeIdCou...	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
QueryFirstCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
QueryNextCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
RegisterNodesCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
UnregisterNodesCount	BrbUaServiceCounterData_TYP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	

Die „ClientDescription“ befindet sich in einer eigenen Unterstruktur:

BrbUaApplicationDescription_TYP			<input checked="" type="checkbox"/>	Applikations-Beschreibung
sApplicationUri	STRING[255]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
sProductUri	STRING[255]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
ApplicationName	UALocalizedText	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
eApplicationType	BrbUaApplicationType_ENUM	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
sGatewayServerUri	STRING[255]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
sDiscoveryProfileUri	STRING[255]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
DiscoveryUrlsLength	UAnoOfElements	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
sDiscoveryUrls	STRING[255][0..9]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	

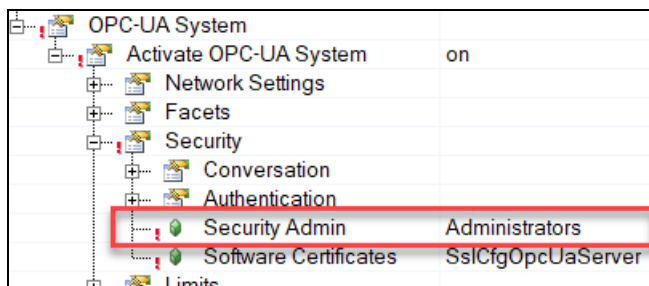
Die meisten Zähler sind ebenfalls als Struktur aufgelegt:

BrbUaServiceCounterData_TYP			<input checked="" type="checkbox"/>	Zähler für Service-Daten
nTotalCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nErrorCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	

Genauere Beschreibung zu den Datenpunkten kann in der OpcUa-Spezifikation Part 5 nachgeschlagen werden.

4.1.11.3 SessionSecurityData

Achtung: Da diese Daten sicherheitsrelevant sind, ist der Zugriff darauf beim B&R-OpcUa-Server nur dann gestattet, wenn dem angemeldeten Benutzer die Rolle zugewiesen ist, welche in der Server-Konfiguration als „Security Admin“ angegeben ist. Beispiel:



Beim Lesen/Abonnieren des Knotens

/Root/Objects/Server/ServerDiagnostics/SessionsDiagnosticsSummary/SessionSecurityDiagnostics
Array

= "ns=0;i=3708"

werden Daten gesendet, die in einer Instanz dieser Struktur abgebildet werden können:

BrbUaSessSecDiagArray_TYP				
Length	UAArrayLength	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Array für Security-Session-Diagnose
Data	BrbUaSessSecDiagData_TYP[0..nBRBUA_SRVDIAG_SESSION_INDEX_MAX]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	

Es handelt sich dabei um ein dynamisches Array (siehe auch [Hinweise zu dynamischen Arrays](#)), welches maximal 10 Elemente aufnehmen kann.

Achtung: Sind am Server mehr als 10 Sessions aktiv, kann das empfangene Array nicht auf die Variable kopiert werden. Die Variable wird dann nicht mehr aktualisiert!

Müssen mehr Sessions abgebildet werden können, muss ein eigener Datentyp mit einem größeren Array definiert werden.

Ein Element beschreibt die Sicherheits-Einstellungen einer Session:

BrbUaSessSecDiagData_TYP					Security-Diagnose-Daten einer Session
SessionId	UANodeID	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
sClientUserIdOfSession	STRING[64]	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
ClientUserIdHistoryCount	UAnoOfElements	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
sClientUserIdHistory	STRING[10][0..nBRBUA_CLTDIAG_USERHIST_IDX_MAX]	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
sAuthenticationMechanism	STRING[64]	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
sEncoding	STRING[64]	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
sTransportProtocol	STRING[64]	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
eSecurityMode	BrbUaMessageSecurityMode_ENUM	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
sSecurityPolicyUri	STRING[64]	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
ClientCertificate	USINT[0..nBRBUA_CERT_ARRAY_INDEX_MAX]	<input type="checkbox"/>	<input checked="" type="checkbox"/>		

Hinweise zu den einzelnen Datenpunkten:

„SessionId“ enthält den Verweis auf die zugehörige Session.

„ClientUserIdOfSession“ enthält den Namen des aktuell angemeldeten Benutzers, ist aber nur gültig, wenn als Anmeldemechanismus „UserName“ angegeben ist (siehe unten).

„ClientUserIdHistoryCount“ gibt die Anzahl der gültigen Einträge der nachfolgenden Liste an.

„ClientUserIdHistory“ enthält eine Liste der letzten 5 angemeldeten Benutzer, ist aber nur gültig, wenn als Anmeldemechanismus „UserName“ angegeben ist (siehe unten). Die Anmeldung als Anonymous taucht also nicht in der Liste auf.

Hinweis: Der B&R-Server behält nur die letzten 5 Benutzer, also können auch nicht mehr ausgelesen werden. Wird der Benutzer nochmals umgemeldet, schiebt sich die Liste nach oben und der aktuelle Benutzer wird an letzter Stelle eingetragen.

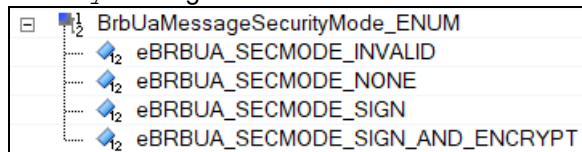
„sAuthenticationMechanism“ enthält den Anmeldemechanismus des aktuellen Benutzers, z.B.

„Anonymous“ oder „UserName“.

„sEncoding“ enthält die Kodierung, z.B. „XML“, „JSON“ oder „UA Binary“.

„sTransportProtocol“ enthält das Protokoll, z.B. „TCP“.

„sSecurityMode“ gibt den verwendeten Sicherheits-Modus mithilfe einer Enumeration an:



„sSecurityPolicyUri“ gibt die verwendete Sicherheits-Policy an, z.B.

„http://opcfoundation.org/UA/SecurityPolicy#Basic256Sha256“.

„ClientCertificate“ enthält das öffentliche Client-Zertifikat als Byte-Array.

Achtung: Das Array wurde mit 2000 Bytes definiert. Werden längere Zertifikate verwendet und kann es deshalb nicht in die Variable eingetragen werden, werden die kompletten „SessionSecurityData“ nicht befüllt! Es sollte dann ein eigener Datentyp mit größerem Array angelegt werden (siehe auch [Hinweise zu dynamischen Arrays](#)).

Genauere Beschreibung zu den Datenpunkten kann in der OpcUa-Spezifikation Part 5 nachgeschlagen werden.

4.1.11.4 SubscriptionDiagData

Beim Lesen/Abonnieren des Knotens

Root/Objects/Server/ServerDiagnostics/SubscriptionsDiagnosticsArray

= “ns=0;i=2290”

werden Daten gesendet, die in einer Instanz dieser Struktur abgebildet werden können:

BrbUaSubscriptDiagArray_TYP	UAArrLength	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Array für Subscription-Diagnose
Length	BrbUaSubscriptDiagData_TYP[0..nBRBUA_SRVDIAG_SUBSCR_INDEX_MAX]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Data		<input type="checkbox"/>	<input checked="" type="checkbox"/>	

Es handelt sich dabei um ein dynamisches Array (siehe auch [Hinweise zu dynamischen Arrays](#)), welches maximal 30 Elemente aufnehmen kann.

Achtung: Sind am Server mehr als 30 Subscriptions aktiv, kann das empfangene Array nicht auf die Variable kopiert werden. Die Variable wird dann nicht mehr aktualisiert!

Müssen mehr Subscriptions abgebildet werden können, muss ein eigener Datentyp mit einem größeren Array definiert werden.

Ein Element beschreibt eine Subscription:

BrbUaSubscriptDiagData_TYP	UAArrLength	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Diagnose-Daten einer Subscription
SessionId	UAArrLength	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nSubscriptionId	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nPriority	USINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
rPublishingInterval	LREAL	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nMaxKeepAliveCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nMaxLifetimeCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nMaxNotificationsPerPublish	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
bPublishingEnabled	BOOL	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nModifyCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nEnableCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nDisableCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nRepublishRequestCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nRepublishMessageRequestCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nTransferRequestCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nTransferredToAltClientCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nTransferredToSameClientCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nPublishRequestCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nDataChangeNotificationsCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nEventNotificationsCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nNotificationsCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nLatePublishRequestCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nCurrentKeepAliveCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nCurrentLifetimeCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nUnacknowledgedMessageCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nDiscardedMessageCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nMonitoredItemCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nDisabledMonitoredItemCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nMonitoringQueueOverflowCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nNextSequenceNumber	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
nEventQueueOverflowCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	

Die Subscriptions werden im Speicher des Servers unabhängig von den Sessions gehalten (daher auch die getrennten Diagnose-Daten). Über den Datenpunkt „`SessionId`“ kann die zugehörige Session herausgefunden werden.

Genauere Beschreibung zu den Datenpunkten kann in der OpcUa-Spezifikation Part 5 nachgeschlagen werden.

4.1.11.5 Hinweise zu den Diagnose-Daten

Um die Diagnose-Daten einmalig zu erhalten, genügt ein Read des jeweiligen Knotens.

Um immer den aktuellen Stand der Daten zu haben, sollten die entsprechenden Knoten in einer Subscription abonniert werden. **Auf keinen Fall sollten sie zyklisch gelesen werden, weil sich dies sehr ungünstig auf die Performance auswirkt!**

Es wird also immer eine Instanz des Datentyps benötigt, deren Variablen-Name beim Lesen/Abonnieren angegeben werden kann und auf den die empfangenen Daten zur applikativen Auswertung kopiert werden.

Da die verschiedenen Datentypen teilweise viel Speicher benötigen, empfiehlt es sich, nur die tatsächlich erforderlichen Daten zu holen. So sind die Daten von [SessionSecurityData](#), welche aufgrund des Zertifikat-Arrays sehr viel Arbeitsspeicher verbrauchen, normalerweise nicht nötig.

Außerdem müssen die abonnierten Daten ja auch übers Netzwerk übertragen werden, was sich auch ungünstig auf dessen Auslastung auswirkt. Deshalb ist auch auf ein entsprechend hohes PublishingInterval der Subscription zu achten ($\geq 1..10$ s).

Wichtig: Einige der Daten werden als dynamisches Array übertragen (z.B. [SessionDiagData](#)). In der IEC-Deklaration der SPS muss das Array aber mit einer festen Größe definiert werden. Ist das empfangene Array größer als das definierte, wird es nicht in die Array-Variable übertragen. In diesem Fall muss das IEC-Array größer definiert werden!

Zu den benötigten Daten hier ein paar Anwendungs-Beispiele:

4.1.11.5.1 Ermitteln der Client-Anzahl

Wenn nur die Anzahl der aktuell mit dem Server verbundenen Clients benötigt wird, genügt das Abonnieren des einzelnen Knotens

```
Root/Objects/Server/ServerDiagnostics/ServerDiagnosticsSummary/CurrentSessionCount  
= "ns=0;i=2277"
```

welcher auch in der Struktur [ServerDiagData](#) vorhanden ist.

4.1.11.5.2 Ermitteln der Clients

Wenn die Information gebraucht wird, welche Clients mit dem Server verbunden sind, so kann dies mit dem Struktur-Array [SessionDiagData](#) ermittelt werden.

Zum Unterscheiden der Clients können die Datenpunkte „`SessionName`“ bzw. „`ClientDescription`“ verwendet werden. Dies setzt natürlich voraus, dass die Clients bei der Anmeldung unterscheidbare Informationen übermitteln. Der Hostname bzw. die IP-Adresse des Clients kann leider nicht ermittelt werden!

Eine eindeutige, aber nur temporär vergebene und nur lokal gültige Unterscheidung kann über den Datenpunkt „`SessionId`“ gemacht werden (wird vom Server vergeben).

4.1.11.5.3 Aktive und inaktive Clients

Wenn sich ein Client korrekt abmeldet, werden die Diagnose-Daten am Server sofort aktualisiert.

Wenn jedoch nur die Verbindung abbricht, wird der Client weiterhin in den Diagnose-Daten aufgeführt. Er wird erst entfernt, wenn die Verbindung nicht innerhalb des SessionTimeout (welcher vom Client bei der Anmeldung mitgegeben wird) wieder aufgenommen wird.

Das bedeutet, dass ein in den Diagnose-Daten enthaltener Client nicht zwangsweise auch eine korrekte Verbindung unterhält.

Um herauszufinden, ob ein Client tatsächlich aktiv ist, kann der Datenpunkt „`dtClientLastContactTime`“ in [SessionDiagData](#) verwendet werden. Solange er sich ändert, ist der Client aktiv.

Dazu sollte beachtet werden, dass dieser Datenpunkt nur dann aktualisiert wird, wenn Daten an den Client gesendet werden. Eine inaktive Verbindung kann detektiert werden, indem die aktuelle Server-Zeit (UTC) mit der des letzten Kontakts (ebenfalls UTC) verglichen wird:

```
bSessionInactive = (ServerStatus.dtCurrentTime - SessionDiagData.dtClientLastContactTime > x)
```

Es wird also berechnet, wie viele Sekunden keine Daten an den Client gesendet wurden. Ist die berechnete Zeit größer als x Sekunden, kann der Client als inaktiv angesehen werden.

Die Anzahl der Sekunden muss natürlich auf den zyklischen Kontakt des Clients abgestimmt sein. Dazu zählt auch das „Lebenszeichen“, das im Intervall des beim Connect des Clients angegebenen Parameters ‚MonitorConnection‘ (manchmal auch ‚KeepAliveInterval‘) vom Client gesendet und vom Server beantwortet wird.

4.1.12 Additional

In diesem Paket finden sich Funktionen, die in kein anderes Paket passen.

4.1.12.1 BrbUaGetRandomGuidIdString

```
plcdword BrbUaGetRandomGuidIdString(BrUaGuidIdString* pGuidIdString, unsigned long nGuidIdStringSize,
plcbit bWithSeparator)
```

Argumente:

BrUaGuidIdString* pGuidIdString
 Zeiger auf den GuidString
UDINT nGuidIdStringSize
 Größe des GuidStrings
BOOL bWithSeparator
 1 = Trennzeichen ‚-‘ wird eingefügt

Rückgabe:

DWORD
0x00000000 = Good (Kein Fehler)
0x80460000 = Bad_StructureMissing (Nullpointer)

Beschreibung:

Erzeugt einen zufälligen GuidString im Format 12345678-1234-1234-1234-123456789012. Er kann zum Test verwendet werden. Die Trennzeichen können optional auch weggelassen werden.

Beispiel:

```
7fdf25f9-7815-e783-488b-a4247f55b88e
cf42e9b5343d9d615884fac8adc92107
```

4.1.12.2 BrbUaGetRandomXmlElement

```
plcdword BrbUaGetRandomXmlElement(BrUaXmlElement* pXmlElement)
```

Argumente:

BrUaXmlElement* pXmlElement
 Zeiger auf das Xml-Element

Rückgabe:

DWORD
0x00000000 = Good (Kein Fehler)
0x80460000 = Bad_StructureMissing (Nullpointer)

Beschreibung:

Erzeugt ein zufälliges Xml-Element mit dem Format <Tag Attribute="x">Value</Tag>, wobei x zwischen 0 und 255 liegt. Sie kann zum Test verwendet werden.

4.2 Client

In diesem Paket finden sich Datentypen und Funktionen für einen Client.

4.2.1 Connection

In diesem Paket finden sich Datentypen und Funktionen für eine Verbindung.

4.2.1.1 BrbUaGetConnectionStatusText

```
plcdword BrbUaGetConnectionStatusText(enum UAConnectionStatus eConnectionStatus, plcstring*
pStatusText, unsigned long nStatusTextSize)
```

Argumente:

```
enum UAConnectionStatus eConnectionStatus
    ClientConnection-Status
STRING* pStatusText
    Zeiger auf den String, der gefüllt werden soll
UDINT nStatusTextSize
    Größe des Strings, der gefüllt werden soll
```

Rückgabe:

```
DWORD
    0x00000000 = Good          (Kein Fehler)
    0x80460000 = Bad_StructureMissing (Nullpointer)
```

Beschreibung:

Die Funktion gibt den symbolischen Text eines Client-Verbindungs-Status zurück.

```
UACS_Connected      = "Connected"
UACS_ConnectionError = "ConnectionError"
UACS_Shutdown       = "Shutdown"
```

4.2.2 RunClient

Diese Sammlung an FB's und Funktionen kapselt einen Library-Client. Damit lässt sich ohne großen applikativen Aufwand ein Client betreiben, welcher über sehr viel automatische Funktionalität verfügt.

Der RunClient kann:

- Namespace-Indizes aufgrund von Namespace-Uri's ermitteln
- Node-Handles für den applikativen Gebrauch ermitteln
- Vordefinierte Datenpunkt-Blöcke lesen
- Vordefinierte Datenpunkt-Blöcke schreiben
- Methoden-Handles für den applikativen Aufruf ermitteln
- Vordefinierte Methoden aufrufen
- Beliebig viele Subscriptions mit beliebig vielen MonitoredItems und/oder EventItems anlegen
- Den Verbindungs-Status liefern

Die meisten Angaben werden vom Anwender statisch in einem Datenobjekt hinterlegt. Nur die Verbindungs-Parameter zum Server können optional auch zur Laufzeit angegeben werden. So kann z.B. die IP-Adresse des Servers aus Maschinen-Parametern stammen, um unterschiedliche Netzwerk-Adressen bei gleicher Sps-SW zu unterstützen.

Die zum RunClient-Betrieb benötigten Daten werden in intern allokierten Speichern gehalten. Der Speicherverbrauch ist optimiert, d.h. es wird nur der benötigte Speicher angefordert.

Über Funktionen erhält der Anwender lesenden Zugriff auf alle internen Daten wie z.B. Fehler-Codes oder Handles, welche dann applikativ weiterverwendet werden können. So ist es z.B. möglich, die Datenblock-Schreib-Funktion nicht zu nutzen, dafür aber mit den ermittelten Node-Handles eine eigene Schreib-Funktion zu implementieren.

4.2.2.1 Allgemeines

Der RunClient besteht aus mehreren Teilen.

Das Datenobjekt muss im AS ausgefüllt werden. Es enthält die Parameter für die verschiedenen Funktionalitäten (siehe unten).

Der FB BrbUaRunClientInit muss im Init-Teil des Tasks aufgerufen werden. Er liest das Datenobjekt aus, allokiert den benötigten Speicher und speichert darin die ausgelesenen Werte.

Der FB BrbUaRunClientCyclic muss im Cyclic-Teil des Tasks aufgerufen werden. Er baut die Verbindung zum Server auf und ermittelt Handles bzw. legt Subscriptions an.

Der FB BrbUaRunClientExit muss im Exit-Teil des Tasks aufgerufen werden. Er gibt den allokierten Speicher wieder frei.

Eine zentrale Struktur-Variable enthält die gesamten Daten zum Betrieb des RunClients. Sie wird beim Aufruf jedes FB's als Zeiger übergeben.

4.2.2.2 Performance und Speicher-Verbrauch

Der RunClient ist sehr performant gestaltet und es sollte eigentlich nicht zu Zykluszeit-Verletzungen kommen. Mit der Anzahl der Funktionalitäten/Datenpunkte steigt allerdings auch die benötigte CPU-Last. So ist vom Anwender zu ermitteln, in welcher Task-Klasse/Zykluszeit der RunClient-Task ausreichend schnell lauffähig ist.

Die empfohlene Taskklasse ist #8. Es kann je nach Zielsystem und Anforderung aber auch eine andere Taskklasse gewählt werden.

Die empfohlene Zykluszeit ist 10ms. Es kann je nach Zielsystem und Anforderung aber auch eine andere Zykluszeit gewählt werden.

Achtung: Es ist darauf zu achten, ein Zielsystem mit genügend CPU-Leistung für die vom Anwender vorgegebenen Funktionalitäten zu verwenden!

Auch der Speicher-Verbrauch ist optimiert. So wird nur der Speicher allokiert, der auch benötigt wird. Bestimmte Funktionalitäten benötigen auch einiges an Arbeits-Speicher. Er wird mittels der Funktion `TMP_alloc` der Bibliothek `SYS_Lib` allokiert und liegt daher im System-RAM.

Achtung: Es ist darauf zu achten, ein Zielsystem mit genügend Arbeits-Speicher für die vom Anwender vorgegebenen Funktionalitäten zu verwenden!

4.2.2.3 Datenobjekt

Das Datenobjekt kann einen beliebigen Namen mit **max. 10 Zeichen** tragen. Es wird im AS vom Anwender befüllt und beinhaltet sämtliche Werte für die auszuführenden Funktionalitäten. Es wird im AS automatisch mitkompiliert und als Binär-Datei auf das Zielsystem übertragen (natürlich nur, wenn es der SW-Configuration unter 'DataObjects' zugewiesen wurde).

Achtung: Wird nur das Datenobjekt geändert, aber nicht der RunClient-Task, muss ein Warmstart ausgeführt werden, da das Datenobjekt nur im Init ausgelesen wird!

4.2.2.3.1 Allgemeines

Um die fehlerfreie Funktionalität des RunClients zu gewährleisten, muss beim Ausfüllen des Datenobjekts folgendes beachtet werden:

- Die unten beschriebene Syntax muss unbedingt eingehalten werden
- Kommentare beginnen mit einem Semikolon ;
- Kommentare werden nicht kompiliert
- Ein einzelner Eintrag muss immer in einer Zeile stehen
- Ein einzelner Eintrag muss zwischen zwei Hoch-Komma „“ eingeschlossen sein
- Ein einzelner Eintrag beginnt immer mit einem Schlüsselwort, gefolgt von den Parametern
- Ein Schlüsselwort beginnt immer mit dem Paragraphen-Zeichen § und endet mit :
- Die Reihenfolge der einzelnen Einträge ist in vielen Fällen entscheidend (siehe unten)
- Leerzeilen zwischen den Einträgen werden ignoriert
- Die Parameter in einem Eintrag müssen durch ein Komma , getrennt sein
- Ein Parameter-Name endet immer mit einem =
- Die Reihenfolge der Parameter innerhalb eines Eintrags muss **unbedingt** eingehalten werden
- Leerzeichen/Tabs zwischen den Parameter werden ignoriert

Die einzelnen Einträge sind optional. Eingetragen sollte nur das sein, was die Anwendung auch benötigt. Nachfolgend werden nur die Syntax und die Bedeutung der Schlüsselwörter/Parameter erklärt. Der Zugriff auf die eingelesenen bzw. ermittelten Werte zur applikativen Verwendung folgt weiter unten.

Einige der Parameter eines Eintrags können auch optional sein. Sie werden dann mit einem Default-Wert besetzt.

Die Syntax von anzugebenden Variablen-Namen entspricht der Angabe

<AppModul>::<Task>:<Variable>.<Element>

Diese ist auch in der AS-Hilfe beschrieben (GUID = 28c9e872-0274-444d-8b4d-0aefb5bad3f6).

4.2.2.3.2 Connection

Dieser erste Eintrag für die Verbindungs-Parameter lässt dem Anwender drei Optionen:

1. Der Eintrag enthält alle Verbindungs-Parameter.
2. Der Eintrag wird weggelassen. Alle Verbindungs-Parameter müssen dann vor dem ersten Connect-Kommando im Cyclic-FB (siehe unten) im Programm gesetzt werden.
3. Der Eintrag enthält nur einige der Verbindungs-Parameter, die restlichen werden im Programm gesetzt.

Mit diesem Konzept können Verbindungs-Parameter (z.B. die Ip-Adresse und der Port des Servers in der Endpoint-Url) auch zur Laufzeit bestimmt werden. Dies lässt die Möglichkeit offen, dasselbe Datenobjekt für verschiedene Maschinen in verschiedenen Netzwerken zu verwenden.

Unabhängig davon, ob der Eintrag vorhanden ist oder nicht, werden alle Parameter vor dem Einlesen des Datenobjekts im Init-FB mit Standard-Werten besetzt (siehe Tabelle). So brauchen manche Parameter vom Anwender gar nicht besetzt werden.

Die Syntax ist folgende:

```
"$CONNECTION: Endpoint=ServerUrl, SessionName=Name, AppName=Name, SecPolicy=Policy, SecMode=Mode, CertificateStore=CfgName, ServerUri=Uri, CheckSrvCert=0, UserToken=IdentityTokenType, TokenPar1=UserName, TokenPar2=UserPassword, VendorPar=SpecificPar, SessionTimeout=Timeout, Monitor=LifeTimeInterval, Locales=LanguageList, ConnectTimeout=Timeout, AccessTimeout=Timeout"
```

Beispiel:

```
"$CONNECTION: Endpoint=opc.tcp://127.0.0.1:4840, SessionName=BrbUaClient, CertStore=SslCfgOpcUaClient, UserToken=1, TokenPar1=Admin, TokenPar2=admin, Locales=de,en"
```

Parameter	Beschreibung	Pflicht	Default-Wert
Endpoint=	Endpoint des Servers im Format opc.tcp://<hostname>:<port> Statt des Hostnames kann natürlich auch eine Ip-Adresse verwendet werden	Nein	Leer
SessionName=	Name der Sitzung	Nein	Leer
AppName=	Name der Client-Anwendung	Nein	Leer
SecPolicy=	Der Wert der Sicherheits-Richtlinien-Enumeration (siehe AS-Hilfe UASecurityPolicy) 0=UASecurityPolicy_BestAvailable 1=UASecurityPolicy_None 2=UASecurityPolicy_Basic128Rsa15 3=UASecurityPolicy_Basic256 4=UASecurityPolicy_Basic256Sha256	Nein	0 = UASecurityPolicy_BestAvailable
SecMode=	Der Wert der Sicherheits-Modus-Enumeration (siehe AS-Hilfe UASecurityMsgMode) 0=UASecurityMsgMode_BestAvailable 1=UASecurityMsgMode_None 2=UASecurityMsgMode_Sign 3=UASecurityMsgMode_SignEncrypt	Nein	0 = UASecurityMsgMode_BestAvailable
CertStore=	Name der optionalen SSL-Konfiguration	Nein	Leer
ServerUri=	Optionale Server-Uri	Nein	Leer
CheckSrvCert=	Trust-List-Prüfung des Server-Zertifikats	Nein	0
UserToken=	Der Wert der Identitäts-Token-Enumeration (siehe AS-Hilfe UAUserIdentityToken) 0=UAUITT_Anonymous 1=UAUITT_Username	Nein	0 = UAUITT_Anonymous
TokenPar1=	Benutzer-Name bei UAUITT_Username	Nein	Leer
TokenPar2=	Benutzer-Passwort bei UAUITT_Username	Nein	Leer
VendorPar=	Client-spezifische Parameter	Nein	Leer
SessionTimeout=	Server-Sitzungs-Timeout in [ms]	Nein	10000
Monitor=	Überwachungs-Intervall in [ms]	Nein	2000
Locales=	Angabe der gewünschten Sprachen (max. 5).	Nein	en

	Die einzelnen Elemente werden mit ; getrennt		
ConnectTimeout=	Timeout in [ms] für den Verbindungs-Aufbau	Nein	3000
AccessTimeout=	Timeout in [ms] für alle späteren Zugriffe des Cyclic-FB (Read, Write usw.)	Nein	5000

Die Parameter entsprechen der Beschreibung in der AS-Hilfe zum FB `AsOpcUac.UA_Connect`.

Hinweis: Die Angabe `TransportProfile` wird intern immer auf `UATP_UATcp` gesetzt, weil dieses Profil als einziges unterstützt wird.

Achtung: Werden der Benutzer-Name und das Benutzer-Passwort hier angegeben, sind sie als Klartext im Datenobjekt hinterlegt!

Der Parameter `AccessTimeout` wird für alle späteren Zugriffe auf den Server benutzt, also z.B. Ermitteln von Handles, Lesen, Schreiben, Erzeugen von Subscriptions, Anmeldung von MonitoredItems usw.

4.2.2.3.3 Namespaces

Nach dem Verbindungs-Aufbau können für beliebig viele Namespace-Uri's die Namespace-Indizes ermittelt werden.

Diese Einträge sollten noch vor den folgenden Einträgen im Datenobjekt stehen, da folgende Einträge u.U. von dieser Tabelle Gebrauch machen (siehe unten).

Die Syntax ist folgende:

```
"$NAMESPACE: Uri=NamespaceUri"
```

Beispiele:

```
"$NAMESPACE: Uri=http://opcfoundation.org/UA/"
"$NAMESPACE: Uri=http://opcfoundation.org/UA/DI/"
"$NAMESPACE: Uri=http://br-automation.com/OpcUa/PLC/"
"$NAMESPACE: Uri=http://br-automation.com/OpcUa/PLC/PV/"
```

Parameter	Beschreibung	Pflicht	Default-Wert
Uri=	Namespace-Uri, zu welchem ein Server-Namespace-Index ermittelt werden soll.	Ja	

Die Parameter entsprechen der Beschreibung in der AS-Hilfe zum FB

`AsOpcUac.UA_GetNamespaceIndex`.

Namespace-Indizes, welche im Datenobjekt bei folgenden Einträgen verwendet werden, beziehen sich auf die Reihenfolge dieser 0-basierten Liste. Muss z.B. später ein Node aus dem B&R-PV-Namespace angegeben werden, so wäre es hier der Index 3, weil er als vierter Eintrag angegeben ist. Der Index wird dann zur Laufzeit mit dem am Server gültigen Index ersetzt. Wird ein Index verwendet, der im Datenmodul nicht angegeben ist, so wird er mit 0 ersetzt.

4.2.2.3.4 NodeHandles

Bei sehr speziellen Anforderungen der Applikation entspricht die Implementierung des RunClients eventuell nicht dem, was der Anwender braucht. In diesem Fall kann sich der Anwender hier die benötigten Node-Handles ermitteln lassen und nach dem Verbindungs-Aufbau für eigene, applikative Implementierungen (z.B. Lesen oder Schreiben) verwenden.

Die Syntax ist folgende:

```
"$NODEHANDLE: Ns=DatObjNamespaceIndex, Id=NodeIdentifier"
```

Beispiele:

```
"$NODEHANDLE: Ns=3, Id=:BrbUaSrvC:VarsLocal.ReadOnly.nUdint"
"$NODEHANDLE: Ns=3, Id=:BrbUaSrvC:VarsLocal.ReadOnly.nDint"
```

Parameter	Beschreibung	Pflicht	Default-Wert
Ns=	Der Datenobjekt-Namespace-Index des Nodes (siehe Namespaces oben).	Ja	
Id=	Der NodeIdentifier des Nodes. Es wird automatisch erkannt, ob es sich um eine Numeric- oder String-Adressierung handelt.	Ja	

Die Parameter entsprechen der Beschreibung in der AS-Hilfe zum FB

`AsOpcUac.UA_NodeGetHandleList`.

4.2.2.3.5 ReadBlocks

Hier ist es möglich, mehrere (max. 65535) Datenpunkte zu einem Block zusammenzustellen, welche später ohne viel Aufwand auf einmal gelesen werden können.

Es sind beliebig viele Blöcke möglich. Hier wird nur die Syntax eines Blockes erklärt. Werden mehrere Blöcke benötigt, müssen sie nur nacheinander im Datenobjekt eingetragen sein.

Die Syntax ist folgende:

```
"$READBLOCK:"  
"$READITEM: Ns=DatObjNamespaceIndex, Id=NodeIdentifizier, Var=VariablenName, AttId=AttributeId"
```

Beispiel:

```
"$READBLOCK:"  
"$READITEM: Ns=3, Id=:BrbUaSrvC:VarsLocal.ReadOnly.nUdint, Var=:BrbUaCltC:NodeClass, AttId=2"  
"$READITEM: Ns=3, Id=:BrbUaSrvC:VarsLocal.ReadOnly.nDint, Var=:BrbUaCltC:ClientVarsRead.nDint"  
"$READITEM: Ns=3, Id=:BrbUaSrvC:VarsLocal.ReadOnly.anInt, Var=:BrbUaCltC:ClientVarsRead.anInt"
```

Der ReadBlock hat keine Parameter, nur die ReadItems.

Parameter	Beschreibung	Pflicht	Default-Wert
Ns=	Der Datenobjekt-Namespace-Index des Nodes (siehe Namespaces oben).	Ja	
Id=	Der NodeIdentifizier des Nodes. Es wird automatisch erkannt, ob es sich um eine Numeric- oder String-Adressierung handelt.	Ja	
Var=	Der Name der Variablen, auf die der gelesene Wert geschrieben wird.	Ja	
AttId=	Die Id des Attributs, das adressiert wird (siehe AS-Hilfe UAAttributeID).	Nein	13 = Value

Die Parameter entsprechen der Beschreibung in der AS-Hilfe zum FB AsOpcUac.UaClt_ReadBulk.

Die angegebene Variable muss natürlich dem Datentyp des gelesenen Wertes entsprechen.

4.2.2.3.6 WriteBlocks

Hier ist es möglich, mehrere (max. 65535) Datenpunkte zu einem Block zusammenzustellen, welche später ohne viel Aufwand auf einmal geschrieben werden können.

Es sind beliebig viele Blöcke möglich. Hier wird nur die Syntax eines Blockes erklärt. Werden mehrere Blöcke benötigt, müssen sie nur nacheinander im Datenobjekt eingetragen sein.

Die Syntax ist folgende:

```
"$WRITEBLOCK:"  
"$WRITEITEM: Ns=DatObjNamespaceIndex, Id=NodeIdentifizier, Var=VariablenName, AttId=AttributeId"
```

Beispiel:

```
"$WRITEBLOCK:"  
"$WRITEITEM: Ns=3, Id=:BrbUaSrvC:VarsLocal.WriteOnly.nUdint, Var=:BrbUaCltC:ClientVarsRead.NodeClass, AttId=2"  
"$WRITEITEM: Ns=3, Id=:BrbUaSrvC:VarsLocal.WriteOnly.nDint, Var=:BrbUaCltC:ClientVarsRead.nDint"  
"$WRITEITEM: Ns=3, Id=:BrbUaSrvC:VarsLocal.WriteOnly.anInt, Var=:BrbUaCltC:ClientVarsRead.anInt"
```

Der WriteBlock hat keine Parameter, nur die WriteItems.

Parameter	Beschreibung	Pflicht	Default-Wert
Ns=	Der Datenobjekt-Namespace-Index des Nodes (siehe Namespaces oben).	Ja	
Id=	Der NodeIdentifizier des Nodes. Es wird automatisch erkannt, ob es sich um eine Numeric- oder String-Adressierung handelt.	Ja	
Var=	Der Name der Variablen, von der der zu schreibende Wert gelesen wird.	Ja	
AttId=	Die Id des Attributs, das adressiert wird (siehe AS-Hilfe UAAttributeID).	Nein	13 = Value

Die Parameter entsprechen der Beschreibung in der AS-Hilfe zum FB AsOpcUac.UaClt_WriteBulk.

Die angegebene Variable muss natürlich dem Datentyp des geschriebenen Wertes entsprechen.

4.2.2.3.7 Methods

Hier ist es möglich, beliebig viele Methoden anzugeben, welche später ohne viel Aufwand aufgerufen werden können. Aufgrund der Beschränkung durch die PlcOpen-Spezifikation sind nur jeweils bis zu 10 Eingangs- bzw. Ausgangs-Argumente möglich.

Hier wird nur die Syntax einer Methode erklärt. Werden mehrere Methoden benötigt, müssen sie nur nacheinander im Datenobjekt eingetragen sein.

Die Syntax ist folgende:

```
"$METHOD: Ns=DatObjNamespaceIndex, ObjectId=ObjectNodeIdentifizier, MethodId=MethodNodeIdentifizier,  
Timeout=Timeout"
```

```
"$METHODARGIN: Name=ArgumentName, Var=VariablenName"  
"$METHODARGOUT: Name=ArgumentName, Var=VariablenName"
```

Beispiel:

```
"$METHOD: Ns=3, ObjectId=:BrbUaSrvC, MethodId=:BrbUaSrvC:Calculate, Timeout=1000"  
"$METHODARGIN: Name=nValue0, Var=:BrbUaClC:ClientMethCalculateArgs.In.nVal0"  
"$METHODARGIN: Name=nValue1, Var=:BrbUaClC:ClientMethCalculateArgs.In.nVal1"  
"$METHODARGIN: Name=nArrayIn, Var=:BrbUaClC:ClientMethCalculateArgs.In.nArray"  
"$METHODARGOUT: Name=nAddition, Var=:BrbUaClC:ClientMethCalculateArgs.Out.nAddition"  
"$METHODARGOUT: Name=nMultiplication, Var=:BrbUaClC:ClientMethCalculateArgs.Out.nMultiplication"  
"$METHODARGOUT: Name=nArrayOut, Var=:BrbUaClC:ClientMethCalculateArgs.Out.nArray"
```

Method

Parameter	Beschreibung	Pflicht	Default-Wert
Ns=	Der Datenobjekt-Namespace-Index des ObjectNodes und des MethodNodes (siehe Namespaces oben).	Ja	
ObjectId=	Der NodeIdentifizier des ObjectNodes. Es wird automatisch erkannt, ob es sich um eine Numeric- oder String-Adressierung handelt.	Ja	
MethodId=	Der NodeIdentifizier des MethodNodes. Es wird automatisch erkannt, ob es sich um eine Numeric- oder String-Adressierung handelt.	Ja	
Timeout=	Der CallTimeout in [ms].	Nein	5000ms

Die Parameter entsprechen der Beschreibung in der AS-Hilfe zum FB

AsOpcUac.UA_MethodGetHandle.

Arguments

Parameter	Beschreibung	Pflicht	Default-Wert
Name=	Der Name des Arguments.	Ja	
Var=	Der Name der Variablen, die den Argument-Wert enthält.	Ja	

Die Parameter entsprechen der Beschreibung in der AS-Hilfe zum FB AsOpcUac.UA_MethodCall.

Die angegebene Variable muss natürlich dem Datentyp des Arguments entsprechen.

Werden für eine Methode, welche eigentlich Argumente hat, keine Argumente angegeben, so kann die Methode später nicht ohne Fehler über diesen RunClient aufgerufen werden. Allerdings wird trotzdem das MethodHandle ermittelt, mit dem der Call applikativ implementiert werden kann.

4.2.2.3.8 Subscription

Hier ist es möglich, beliebig viele Subscriptions anzugeben, welche automatisch angelegt werden. Optional können dann beliebig viele MonitoredItems bzw. EventItems angegeben werden (siehe unten). Hier wird nur die Syntax einer Subscription erklärt. Werden mehrere Subscriptions benötigt, müssen sie nur nacheinander im Datenobjekt eingetragen sein.

Die Syntax ist folgende:

```
"$SUBSCRIPTION: Ena=PublishingEnable, Prio=Priority, PubInt=PublishingInterval"
```

Beispiel:

```
"$SUBSCRIPTION: Ena=1, Prio=200, PubInt=100"
```

Parameter	Beschreibung	Pflicht	Default-Wert
Ena=	Aktiviert die Datenübergabe am Server.	Nein	1
Prio=	Die Priorität der Subscription am Server (0..255: Je größer, desto höher die Priorität).	Nein	128
PubInt=	Zeitintervall in [ms], wie oft die Datenübergabe an den Client erfolgen soll.	Nein	500ms

Die Parameter entsprechen der Beschreibung in der AS-Hilfe zum FB

AsOpcUac.UA_SubscriptionCreate.

4.2.2.3.8.1 MonitoredItem

Unterhalb des Subscriptions-Eintrags ist es möglich, beliebig viele MonitoredItems anzugeben, welche dann in diese Subscription hinzugefügt werden.

Hier wird nur die Syntax eines MonitoredItems erklärt. Werden mehrere MonitoredItems benötigt, müssen sie nur nacheinander im Datenobjekt eingetragen sein.

Die Syntax ist folgende:

```
"$MONITEM: Ns=DatObjNamespaceIndex, Id=NodeIdentifier, Var=VariablenName, AttId=AttributeId, Sampling=SamplingInterval, Queue=QueueSize, Discard=DiscardOldest, DeadType=DeadbandType, Deadband=Deadband"
```

Beispiel:

```
"$SUBSCRIPTION: Ena=1, Prio=200, PubInt=500"  
"$MONITEM: Ns=3, Id::BrbUaSrvC:VarsLocal.ReadOnly.bBool, Var::BrbUaCltC:ClientVarsSubscription.bBool"  
"$MONITEM: Ns=3, Id::BrbUaSrvC:VarsLocal.ReadOnly.nUsint, Var::BrbUaCltC:ClientVarsSubscription.nUsint"  
"$MONITEM: Ns=3, Id::BrbUaSrvC:VarsLocal.ReadOnly.nUint, Var::BrbUaCltC:ClientVarsSubscription.nUint"  
"$MONITEM: Ns=3, Id::BrbUaSrvC:VarsLocal.ReadOnly.nUdint, Var::BrbUaCltC:ClientVarsSubscription.nUdint"
```

Parameter	Beschreibung	Pflicht	Default-Wert
Ns=	Der Datenobjekt-Namespace-Index des Nodes (siehe Namespaces oben).	Ja	
Id=	Der NodeIdentifier des Nodes. Es wird automatisch erkannt, ob es sich um eine Numeric- oder String-Adressierung handelt.	Ja	
Var=	Der Name der Variablen, auf die der empfangene Wert geschrieben wird.	Ja	
AttId=	Die Id des Attributs, das adressiert wird (siehe AS-Hilfe UAAttributeID).	Nein	13 = Value
Sampling=	Intervall in [ms], in dem der Server den Wert auf Änderung überprüft.	Nein	PubInt der Subscription
Queue=	Größe des Wert-Änderungs-Puffers am Server. Achtung: Bei Queue > 0 werden empfangene Werte nicht automatisch auf die Variablen geschrieben (Controller-Sync-Modus)! In diesem Fall muss der Anwender zusätzlich den Baustein UA_MonitoredItemOperate aufrufen (siehe AS-Hilfe zum Parameter QueueSize)!	Nein	0
Discard=	Verhalten bei einem Überlauf des Puffers: 0: Der neueste Eintrag wird verworfen 1: Der älteste Eintrag wird verworfen	Nein	1
DeadType=	Gibt an, ob und wie eine Hysterese angewendet wird (siehe AS-Hilfe UADeadbandType): 0=UADeadbandType_None 1=UADeadbandType_Absolute 2=UADeadbandType_Percent	Nein	0
Deadband=	Wert der Hysterese als REAL	Nein	0.0

Die Parameter entsprechen der Beschreibung in der AS-Hilfe zum FB

AsOpcUac.UA_MonitoredItemAddList.

Hinweis: Werden die optionalen Parameter nicht angegeben, wird das Item in dem zu 99% aller Fälle benötigten Modus betrieben:

- Wert wird im Intervall der Subscription überwacht
- Es wird immer nur der aktuelle Wert übertragen
- Wert-Änderung wird sofort beim Empfang automatisch auf die gemappte Variable übertragen

4.2.2.3.8.2 EventItem + EventField

Unterhalb des Subscriptions-Eintrags ist es möglich, beliebig viele EventItems anzugeben, welche dann in diese Subscription hinzugefügt werden. Bei einem EventItem können bis zu 64 EventFields angegeben werden, welche dann abonniert werden.

Hinweis: Gesendet werden nur Fields, welche sowohl am Client abonniert sowie am Server parametrierbar sind!

Hier wird nur die Syntax eines EventItems bzw. Fields erklärt. Werden mehrere EventItems/Fields benötigt, müssen sie nur nacheinander im Datenobjekt eingetragen sein.

Die Syntax ist folgende:

```
"$EVTITEM: EvtNs=EventDatObjNamespaceIndex, EvtId=EventNodeIdentifier, TypeNs=TypeDatObjNamespaceIndex, TypeId=TypeNodeIdentifier, Timeout=Timeout, CallOperate=OperateAufruf"
```

```
"$EVTFIELD: FieldNs=DatObjNamespaceIndex, FieldPath=FieldName, [FieldPath=,] Var=VariablenName"
```

Beispiele:

```
"$SUBSCRIPTION: Ena=1, Prio=250, PubInt=100"  
"$EVTITEM: EvtNs=0, EvtId=2253, TypeNs=0, TypeId=2311, Timeout=1000, CallOperate=1"  
"$EVTFIELD: FieldNs=0, FieldPath=EventId, Var::BrbUaCltC:ClientReceivedEvent.EventId"  
"$EVTFIELD: FieldNs=7, FieldPath=EventType, Var::BrbUaCltC:ClientReceivedEvent.EventType"  
"$EVTFIELD: FieldNs=0, FieldPath=LocalTime, Var::BrbUaCltC:ClientReceivedEvent.LocalTime"  
"$EVTFIELD: FieldNs=0, FieldPath=Message, Var::BrbUaCltC:ClientReceivedEvent.Message"  
"$EVTFIELD: FieldNs=0, FieldPath=ReceiveTime, Var::BrbUaCltC:ClientReceivedEvent.ReceiveTime"
```



```
"SEVTFIELD: FieldNs=0, FieldPath=Severity, Var=:BrbUaCltC:ClientReceivedEvent.nSeverity"  
"SEVTFIELD: FieldNs=0, FieldPath=SourceName, Var=:BrbUaCltC:ClientReceivedEvent.sSourceName"  
"SEVTFIELD: FieldNs=0, FieldPath=SourceNode, Var=:BrbUaCltC:ClientReceivedEvent.SourceNode"  
"SEVTFIELD: FieldNs=0, FieldPath=Transition, Var=:BrbUaCltC:ClientReceivedEvent.Transition"  
"SEVTFIELD: FieldNs=0, FieldPath=Transition, FieldPath=Id, Var=:BrbUaCltC:ClientReceivedEvent.nTransitionId"
```

EventItem

Parameter	Beschreibung	Pflicht	Default-Wert
EvtNs=	Der Datenobjekt-Namespace-Index des Nodes, an dem das Event abonniert werden soll (siehe Namespaces oben).	Nein	0
EvtId=	Der Nodeldentifizier des Nodes, an dem das Event abonniert werden soll. Meistens ist dies der Server Knoten i=2253. Es wird automatisch erkannt, ob es sich um eine Numeric- oder String-Adressierung handelt.	Nein	2253
TypeNs=	Der Datenobjekt-Namespace-Index des Nodes, der den zu abonnierenden Event-Typen angibt (siehe Namespaces oben).	Nein	0
TypeId=	Der Nodeldentifizier des Nodes, der den zu abonnierenden Event-Typen angibt, z.B. 2311 für den Typ <code>TransitionEventType</code> . Es wird automatisch erkannt, ob es sich um eine Numeric- oder String-Adressierung handelt.	Nein	2041 (=BaseEventType)
Timeout=	Timeout in [ms] für den Aufruf von <code>UA_EventItemAdd</code> .	Nein	5000
CallOperate=	Angabe, ob automatisch ein Aufruf von <code>UA_EventItemOperate</code> erfolgen soll.	Nein	1

Die Parameter entsprechen der Beschreibung in der AS-Hilfe zum FB `AsOpcUac.UA_EventItemAdd`. Hinweis: Gesendete Events werden nur erkannt, wenn zyklisch der Baustein `UA_EventItemOperate` aufgerufen wird. Ist der Parameter `CallOperate` auf 1, übernimmt diese Aufgabe der Cyclic-FB (siehe [Operate-Aufruf von EventItems](#) unten). Ist er 0, so muss der Aufruf vom Anwender erledigt werden.

EventField

Parameter	Beschreibung	Pflicht	Default-Wert
FieldNs=	Der Datenobjekt-Namespace-Index für alle Pfad-Elemente des Fields (siehe Namespaces oben).	Ja	
FieldPath=	Ein Pfad-Element zur Beschreibung des adressierten Fields. Enthält der Pfad mehrere Elemente, muss die Angabe für jedes Element wiederholt werden (max. 16 sind möglich). Die meisten Felder sind durch Angabe 1 Pfad-Elements erreichbar, z.B. <code>Message</code> .	Ja	
Var=	Der Name der Variablen, auf die der empfangene Field-Wert geschrieben wird.	Ja	

Die Parameter entsprechen der Beschreibung in der AS-Hilfe zum FB `AsOpcUac.UA_EventItemAdd`.

4.2.2.4 Funktionsbausteine zum Betrieb des RunClients

Zum Betrieb des RunClients müssen verschiedene FB's aufgerufen werden.

4.2.2.4.1 BrbUaRunClientInit

```
void BrbUaRunClientInit(struct BrbUaRunClientInit* inst)
```

Argumente:

```
struct BrbUaRunClientInit* inst  
Zeiger auf die Funktionsblock-Instanz
```

Eingänge:

```
struct BrbUaRunClient_TYP* pRunClient  
Zeiger auf die RunClient-Struktur (genaue Beschreibung siehe unten)
```

Ausgänge:

```
UINT nStatus  
Funktionsblock-Status  
eBRB_OK  
eBRB_ERR_NULL_POINTER  
eBRB_ERR_BUSY  
1..65534 = Fehlermeldung. Wenn es sich um einen System-Fehler (<50000) handelt, kann die Beschreibung dazu in der Studio-Hilfe gefunden werden.
```

Beschreibung:

Dieser FB **muss** im Init-Teil des Task aufgerufen werden.

Achtung: Wird er im zyklischen Teil aufgerufen, kann es zu Zykluszeit-Verletzung kommen!

Er liest das Datenobjekt aus, allokiert den benötigten Speicher und speichert darin die ausgelesenen Werte.

Wichtig: Da der Init nur einen Zyklus durchlaufen wird, der FB aber asynchron arbeitet, muss er in einer Schleife aufgerufen werden, bis der Status `<> eBRB_ERR_BUSY` meldet.

Folgende Daten müssen vor dem Aufruf in der RunClient-Struktur (genaue Beschreibung siehe unten) gesetzt werden:

`-RunClient.Cfg.sCfgDataObjName` Der Name des Datenobjekts (max. 10 Zeichen)

4.2.2.4.2 BrbUaRunClientCyclic

```
void BrbUaRunClientCyclic(struct BrbUaRunClientCyclic* inst)
```

Argumente:

```
struct BrbUaRunClientCyclic* inst  
    Zeiger auf die Funktionsblock-Instanz
```

Eingänge:

```
struct BrbUaRunClient_TYP* pRunClient  
    Zeiger auf die RunClient-Struktur (genaue Beschreibung siehe unten)
```

Ausgänge:

```
UINT nStatus  
    Funktionsblock-Status  
    eBRB_OK  
    eBRB_ERR_NULL_POINTER  
    eBRB_ERR_INVALID_PARAMETER  
    eBRB_ERR_BUSY  
    1..65534 = Fehlermeldung. Wenn es sich um einen System-Fehler (<50000) handelt, kann die Beschreibung dazu  
    in der Studio-Hilfe gefunden werden.
```

Beschreibung:

Dieser FB **muss** zyklisch aufgerufen werden.

Er verbindet den Client mit dem Server und ermittelt Handles bzw. legt Subscriptions an, erledigt also alle Dinge, die erst nach Verbindungs-Aufbau gemacht werden können oder zyklisch erledigt werden müssen. Über Kommandos kann ein Connect bzw. Disconnect ausgeführt werden.

Wurde vorher nicht der Init-FB aufgerufen, wird der Status `eBRB_ERR_INVALID_PARAMETER` zurückgegeben.

Folgende Daten müssen vor dem Aufruf in der RunClient-Struktur (genaue Beschreibung siehe unten) gesetzt werden, wenn sie nicht im Datenobjekt angegeben sind:

`-RunClient.Cfg.xxx` Die Verbindungs-Parameter

4.2.2.4.2.1 Operate-Aufruf von EventlItems

Ist bei einem oder mehreren EventlItems im Datenobjekt der Parameter `CallOperate` gesetzt, so wird während des States `eBRB_RCCLTSTATE_CONNECTED` für jedes dieser EventlItems der FB `UA_EventItemOperate` aufgerufen und so der Empfang behandelt.

Achtung: Ist der Parameter nicht gesetzt, muss sich der Anwender selbst um das Aufrufen dieses FB's für dieses EventlItem kümmern. Wird der FB nicht aufgerufen, wird der Empfang dieses Events nicht behandelt!

Der Aufruf bei mehreren EventlItems geschieht nach folgendem Konzept:

1. Das nächste EventlItem mit `CallOperate = 1` wird gesucht.
2. Der FB `UA_EventItemOperate` wird für dieses eine Item in jedem Zyklus solange aufgerufen, bis der Baustein Done oder Error meldet.
3. Im nächsten Zyklus wird wieder bei 1. begonnen

Durch das sequentielle Aufrufen ist gewährleistet, dass jedes EventItem behandelt wird, es aber bei sehr vielen EventItems zu keiner Zykluszeit-Verletzung kommt. Sollten dadurch die EventItems zu selten drankommen, muss das Aufrufen applikativ gemacht werden.

4.2.2.4.3 BrbUaRunClientExit

```
void BrbUaRunClientExit(struct BrbUaRunClientExit* inst)
```

Argumente:

```
struct BrbUaRunClientExit* inst  
    Zeiger auf die Funktionsblock-Instanz
```

Eingänge:

```
struct BrbUaRunClient_TYP* pRunClient  
    Zeiger auf die RunClient-Struktur (genaue Beschreibung siehe unten)
```

Ausgänge:

```
UINT nStatus  
    Funktionsblock-Status  
    eBRB_OK  
    eBRB_ERR_NULL_POINTER  
    eBRB_ERR_UA_ERROR  
    eBRB_ERR_BUSY  
    1..65534 = Fehlermeldung. Wenn es sich um einen System-Fehler (<50000) handelt, kann die Beschreibung dazu  
    in der Studio-Hilfe gefunden werden.
```

Beschreibung:

Dieser FB **muss** im Exit-Teil des Task aufgerufen werden.

Achtung: Wird er im zyklischen Teil aufgerufen, kann es zu Zykluszeit-Verletzung kommen!

Er trennt die Verbindung zum Server, setzt alle ermittelten Handles auf 0 und gibt den allokierten Speicher wieder frei. Dies ist besonders wichtig, wenn der Task nach Änderung übertragen wird. Dabei wird zuerst der Exit des alten Tasks und dann der Init des neuen Tasks abgearbeitet (siehe AS-Hilfe).

Würde dieser FB im Task-Exit nicht aufgerufen, bliebe der alte allokierte Speicher bestehen und es würde neuer allokiert, führte also zu einem Speicherleck. Zusätzlich würde am Server ebenfalls die Verbindung bis zum SessionTimeout bestehen bleiben.

Wichtig: Da der Exit nur einen Zyklus durchlaufen wird, der FB aber asynchron arbeitet, muss er in einer Schleife aufgerufen werden, bis der Status \neq eBRB_ERR_BUSY meldet.

4.2.2.5 RunClient-Struktur

Diese Struktur muss vom Anwender angelegt und jedem der FB's als Zeiger übergeben werden.

Sie enthält die gesamten Daten zum Betrieb des RunClients. Manche der Daten müssen vom Anwender vor Aufruf eines FB's gesetzt werden, andere werden automatisch ermittelt. Außerdem können damit Kommandos (z.B. Connect oder Disconnect) übergeben, der Status der Verbindung ausgelesen werden und einiges mehr.

Aus Übersichtlichkeits-Gründen ist die Struktur in einige Unterstrukturen aufgeteilt. Die meisten dieser Unterstrukturen sind interne Daten und dürfen nicht verändert werden!

BrbUaRunClient_TYP		RunClient-Struktur zum Betreiben eines Clients
Cfg	BrbUaRcCfg_TYP	Verbindungs-Parameter
eCmd	BrbUaRcCommandos_ENUM	Client-Kommandos (z.B. Verbindung trennen)
Connection	BrbUaRcConnection_TYP	Daten der Verbindung
Namespaces	BrbUaRcNamespaces_TYP	Daten der Namespaces
NodeHandles	BrbUaRcNodeHandles_TYP	Daten der NodeHandles
ReadBlocks	BrbUaRcReadBlocks_TYP	Daten der ReadBlocks
WriteBlocks	BrbUaRcWriteBlocks_TYP	Daten der WriteBlocks
Methods	BrbUaRcMethods_TYP	Daten der Methods
Subscriptions	BrbUaRcSubscriptions_TYP	Daten der Subscriptions
State	BrbUaRcState_TYP	Status des Clients, der Verbindung und des Servers

4.2.2.5.1 Cfg

BrbUaRcCfg_TYP		Verbindungs-Parameter
sCfgDataObjName	STRING[nBRBUA_DATAOBJECT_NAME_CHAR_MAX]	Name des Datenobjekts
sServerEndpointUrl	STRING[sBRBUA_ENDPOINT_URL_CHAR_MAX]	Endpoint-Url
SessionConnectInfo	UASessionConnectInfo	Angaben zur Verbindung
tConnectTimeout	TIME	Timeout für den Connect in [ms]
tAccessTimeout	TIME	Timeout für Zugriff auf den Server in [ms]

Der erste Parameter `sCfgDataObjName` gibt den Namen des Datenobjekts an (max. 10 Zeichen), in welchem die funktionalen Parameter hinterlegt sind. Er muss vom Anwender schon vor dem Aufruf des Init-FB's korrekt belegt sein.

Alle anderen Parameter sind Verbindungs-Parameter und werden im Init-FB mit Default-Werten vorbelegt (siehe oben) und, wenn vorhanden, aus dem Datenobjekt gelesen. Ist der Eintrag im Datenobjekt nicht oder nur teilweise vorhanden, muss der Anwender die fehlenden Parameter vor dem ersten Connect-Kommando am Cyclic-FB (siehe unten) selbst korrekt besetzen. Somit kann dasselbe Datenobjekt verwendet werden, auch wenn z.B. die IP-Adresse des Servers an der Visu änderbar ist. Die Parameter entsprechen der Beschreibung in der AS-Hilfe zum FB `AsOpcUac.UA_Connect`.

4.2.2.5.2 eCmd

Hier können applikative Kommandos an den RunClient übergeben werden:

BrbUaRcCommandos_ENUM		Client-Kommandos
eBRB_RCCLTCMD_NONE		Aktuell kein Kommando
eBRB_RCCLTCMD_CONNECT		Verbindung aufbauen
eBRB_RCCLTCMD_DISCONNECT		Verbindung trennen

Nach dem Absetzen eines Kommandos wird die Schnittstelle im selben Zyklus wieder auf `eBRB_RCCLTCMD_NONE` gesetzt. Befindet sich der RunClient in einem State, bei welcher der Befehl Sinn ergibt, wird er sofort ausgeführt.

4.2.2.5.2.1 eBRB_RCCLTCMD_CONNECT

Steht `RunClient.State.eClientState` auf `eBRB_RCCLTSTATE_WAIT_FOR_CONNECT`, wird mit diesem Kommando ein Verbindungsaufbau zum Server versucht. Währenddessen steht `RunClient.State.eClientState` auf `eBRB_RCCLTSTATE_CONNECTING`.

Ist dieser nach Ablauf des Timeouts nicht erfolgreich, wird `RunClient.State.eClientState` auf `eBRB_RCCLTSTATE_CONNECT_ERROR` gesetzt. Außerdem ist der entsprechende OpcUa-Status in `RunClient.State.nClientErrordId` enthalten. Ein erneuter Versuch kann mit dem nochmaligen Setzen des Connect-Kommandos gestartet werden (z.B. nach Veränderung der Verbindungs-Parameter).

Ist die Verbindung erfolgreich aufgebaut, werden folgende Aufgaben erledigt:

- Ermitteln der NamespaceIndices
- Ermitteln der NodeHandles
- Ermitteln der ReadItemHandles
- Ermitteln der WriteItemHandles
- Ermitteln der MethodHandles
- Erzeugen der Subscriptions
- Ermitteln der Handles und Anmelden der MonitoredItems
- Ermitteln der Handles und Anmelden der EventItems

Dann geht `RunClient.State.eClientState` auf `eBRB_RCCLTSTATE_CONNECTED`.

Grundsätzlich wird das Connect-Kommando nur in folgende States ausgeführt:

```
eBRB_RCCLTSTATE_WAIT_FOR_CONNECT  
eBRB_RCCLTSTATE_CONNECT_ERROR
```

4.2.2.5.2 eBRB_RCCLTCMD_DISCONNECT

Steht `RunClient.State.eClientState` auf `eBRB_RCCLTSTATE_CONNECTED`, wird mit diesem Kommando die Verbindung getrennt. Währenddessen steht `RunClient.State.eClientState` auf `eBRB_RCCLTSTATE_DISCONNECTING`.

Ist das Trennen nicht erfolgreich (meist, weil der Server gerade nicht erreichbar ist), steht der dazugehörige OpcUa-Status in `RunClient.State.nErrorId`.

Egal, ob erfolgreich oder nicht, es werden dann auf jeden Fall alle ermittelten Handles (von Nodes, Methods, Subscriptions, Monitored- und Event-Items) auf 0 gesetzt, da sie ja dann nicht mehr gültig sind. Danach geht `RunClient.State.eClientState` wieder auf `eBRB_RCCLTSTATE_WAIT_FOR_CONNECT`.

Grundsätzlich wird das Connect-Kommando nur in folgende States ausgeführt:

`eBRB_RCCLTSTATE_CONNECTED`
`eBRB_RCCLTSTATE_CON_INTERRUPTED`

4.2.2.5.3 Connection

BrbUaRcConnection_TYP			Daten der Verbindung
nConnectionHandle	UDINT		Handle der Verbindung
nErrorId	DWORD		OpcUa-Fehler beim Verbinden
nConnectTries	UDINT		Connect-Versuche
nInterruptedCount	UDINT		Anzahl der Verbindungs-Unterbrechungen

`nConnectionHandle`

Der Handle der Verbindung wie vom intern aufgerufenen FB `UA_Connect` zurückgegeben.

`nErrorId`

Der Status wie vom intern aufgerufenen FB `UA_Connect` oder `UA_Disconnect` zurückgegeben.

`nConnectTries`

Die Anzahl der Verbindungs-Versuche. Dieser Zähler wird bei einem Disconnect zurückgesetzt.

`nInterruptedCount`

Die Anzahl der Verbindungs-Unterbrechungen. Dieser Zähler wird bei einem Re-Connect zurückgesetzt.

4.2.2.5.4 Namespaces/NodeHandles/ReadBlocks/WriteBlocks/Methods/Subscriptions

BrbUaRcNamespaces_TYP			Daten der Namespaces
nNamespaceCount	UINT		Anzahl der Namespaces
pNamespaces	UDINT		Interner Zeiger
nMemLen	UDINT		Interne Speichergröße
BrbUaRcNodeHandles_TYP			Daten der NodeHandles
nNodeHandleCount	UINT		Anzahl der NodeHandles
pStart	UDINT		Interner Zeiger
pDatObjNamespaceIndices	UDINT		Interner Zeiger
pNodeIds	UDINT		Interner Zeiger
pNodeHandleErrorIds	UDINT		Interner Zeiger
pNodeHandles	UDINT		Interner Zeiger
nMemLen	UDINT		Interne Speichergröße
BrbUaRcReadBlocks_TYP			Daten der ReadBlocks
nBlockCount	UINT		Anzahl der ReadBlocks
pBlocks	UDINT		Interner Zeiger
nMemLen	UDINT		Interne Speichergröße
BrbUaRcWriteBlocks_TYP			Daten der WriteBlocks
nBlockCount	UINT		Anzahl der WriteBlocks
pBlocks	UDINT		Interner Zeiger
nMemLen	UDINT		Interne Speichergröße

BrbUaRcMethods_TYP		Daten der Methods
nMethodCount	UINT	Anzahl der Methods
pMethods	UDINT	Interner Zeiger
nMemLen	UDINT	Interne Speichergröße
BrbUaRcSubscriptions_TYP		Daten der Subscriptions
nSubscriptionCount	UINT	Anzahl der Subscriptions
pSubscriptions	UDINT	Interner Zeiger
nMemLen	UDINT	Interne Speichergröße
nMonitoredItemsCountTotal	UDINT	Anzahl aller MonitorredItems
nEventItemsCountTotal	UDINT	Anzahl aller EventItems

Diese Unterstrukturen enthalten jeweils die Anzahl der vom Datenobjekt eingelesenen Elemente. Die restlichen Daten darin dienen nur internen Zwecken. Diese Daten dürfen vom Anwender auf keinen Fall verändert werden! Der Anwender-Zugriff auf die intern gehaltenen Daten geschieht über Funktionen (siehe weiter unten).

4.2.2.5.5 State

BrbUaRcState_TYP		Status des Clients und der Verbindung und des Servers
eClientState	BrbUaRcClientStates_ENUM	Status des Clients
nClientErrorId	UDINT	OpcUa-Fehler-Status bei internen Aufrufen
sClientErrorText	STRING[nBRBUA_VALUE_TEXT_CHAR_MAX]	Details zum Fehler
ConnectionStatus	BrbUaRcStateConStatus_TYP	Status des Servers

Hier können verschiedene, vom Cyclic-FB zyklisch ermittelte Stati eingesehen werden.

4.2.2.5.5.1 eClientState

BrbUaRcClientStates_ENUM	
eBRB_RCCLTSTATE_NO_INIT	
eBRB_RCCLTSTATE_INITIALIZING	
eBRB_RCCLTSTATE_INIT_ERROR	
eBRB_RCCLTSTATE_INIT_DONE	
eBRB_RCCLTSTATE_WAIT_FOR_CONNECT	
eBRB_RCCLTSTATE_CONNECTING	
eBRB_RCCLTSTATE_CONNECT_ERROR	
eBRB_RCCLTSTATE_CON_INTERRUPTED	
eBRB_RCCLTSTATE_CONNECTED	
eBRB_RCCLTSTATE_DISCONNECTING	
eBRB_RCCLTSTATE_DISCONNECT_ERROR	
eBRB_RCCLTSTATE_DISCONNECTED	
eBRB_RCCLTSTATE_EXITING	
eBRB_RCCLTSTATE_EXIT_ERROR	
eBRB_RCCLTSTATE_EXIT_DONE	

Dieser Enumerations-Wert stellt die verschiedenen Phasen vom Init über Cyclic bis zum Exit dar. Alle Schritte werden normalerweise durch Aufruf der FB's bzw. Übergabe eines Kommandos vom Anwender angestoßen.

Die Ausnahme bildet der Status `eBRB_RCCLTSTATE_CON_INTERRUPTED`. Er tritt ein, wenn eine Verbindung durch äußerliche Umstände getrennt wird, z.B. durch Kabelbruch, Ausfall eines Hubs/Switches, Beenden des Servers usw. In diesem Fall muss der Anwender nichts unternehmen. Ist der Server wieder erreichbar, werden vom Betriebssystem automatisch alle Funktionalitäten wieder aufgenommen und der Status wechselt wieder auf `eBRB_RCCLTSTATE_CONNECTED`.

Achtung: Es gibt Server anderer Hersteller, die das automatische Einrichten einer temporär verlorenen Verbindung nicht unterstützen, obwohl dies explizit in der OpcUa-Spezifikation gefordert wird. In diesem Fall sollte applikativ ein Disconnect- und anschließend wieder ein Connect-Befehl abgesetzt werden!

4.2.2.5.2 nClientErrorId/sClientErrorText

Tritt beim internen Aufruf eines FB'S der System-Bibliothek AsOpcUac ein Fehler auf, so wird dieser als OpcUa-Fehler-Status auf das Element nClientErrorId übernommen und sClientErrorText enthält einen Klartext, bei welcher Gelegenheit der Fehler auftrat (z.B. beim Ermitteln eines NodeHandles für ein MonitoredItem). Wenn möglich werden auch die 0-basierten Indizes der betroffenen Elemente angegeben (z.B. Subscription#1, MonitoredItem#2). Die Indizes beziehen sich dabei immer auf die Reihenfolge im Datenobjekt.

4.2.2.5.3 ConnectionStatus

BrbUaRcStateConStatus_TYP		Status des Servers
eConnectionStatus	UAConnectionStatus	Status der Verbindung
eServerState	UAServerState	Status des Servers
nServiceLevel	USINT	Level des Servers

Diese Unterstruktur enthält die vom intern aufgerufenen FB UA_ConnectionGetStatus ermittelten Daten über den Verbindungs- bzw. Server-Status (siehe AS-Hilfe).

4.2.2.6 Ausführen von vordefinierten Funktionen

Mit diesen FB's ist es möglich, im Datenobjekt vordefinierte Funktionen auszuführen. Die dabei anzugebenden Indizes beziehen sich auf die Reihenfolge im Datenobjekt. Das erste Element hat den Index 0.

4.2.2.6.1 BrbUaRcReadBlock

```
void BrbUaRcReadBlock(struct BrbUaRcReadBlock* inst)
```

Argumente:

```
struct BrbUaRcReadBlock* inst  
    Zeiger auf die Funktionsblock-Instanz
```

Eingänge:

```
struct BrbUaRunClient_TYP* pRunClient  
    Zeiger auf die RunClient-Struktur (genaue Beschreibung siehe oben)  
UINT nReadBlockIndex  
    Der 0-basierte Index des ReadBlocks laut Reihenfolge im Datenobjekt
```

Ausgänge:

```
UINT nStatus  
    Funktionsblock-Status  
    eBRB_OK  
    eBRB_ERR_NULL_POINTER  
    eBRB_ERR_UA_ERROR  
    eBRB_ERR_UA_NO_ELEMENTS  
    eBRB_ERR_UA_INVALID_INDEX  
    eBRB_ERR_UA_NOT_CONNECTED  
    eBRB_ERR_BUSY  
    1..65534 = Fehlermeldung. Wenn es sich um einen System-Fehler (<50000) handelt, kann die Beschreibung dazu  
    in der Studio-Hilfe gefunden werden.  
UINT nErrorId  
    OpcUa-Fehler-Status  
STRING sErrorId  
    OpcUa-Fehler-Status als Text  
UDINT nReadCount  
    Anzahl der Reads
```

Beschreibung:

Liest alle ReadItems des angegebenen ReadBlocks durch den internen Aufruf von AsOpcUac.UaClt_ReadBulk. Die gelesenen Werte werden auf die gemappten Variablen übertragen.

4.2.2.6.2 BrbUaRcWriteBlock

```
void BrbUaRcWriteBlock(struct BrbUaRcWriteBlock* inst)
```

Argumente:

```
struct BrbUaRcWriteBlock* inst  
    Zeiger auf die Funktionsblock-Instanz
```

Eingänge:

```
struct BrbUaRunClient_TYP* pRunClient  
    Zeiger auf die RunClient-Struktur (genaue Beschreibung siehe oben)  
UINT nWriteBlockIndex  
    Der 0-basierte Index des WriteBlocks laut Reihenfolge im Datenobjekt
```

Ausgänge:

```
UINT nStatus  
    Funktionsblock-Status  
    eBRB_OK  
    eBRB_ERR_NULL_POINTER  
    eBRB_ERR_UA_ERROR  
    eBRB_ERR_UA_NO_ELEMENTS  
    eBRB_ERR_UA_INVALID_INDEX  
    eBRB_ERR_UA_NOT_CONNECTED  
    eBRB_ERR_BUSY  
    1..65534 = Fehlermeldung. Wenn es sich um einen System-Fehler (<50000) handelt, kann die Beschreibung dazu  
    in der Studio-Hilfe gefunden werden.  
UINT nErrorId  
    OpcUa-Fehler-Status  
STRING sErrorId  
    OpcUa-Fehler-Status als Text  
UDINT nWriteCount  
    Anzahl der Writes
```

Beschreibung:

Schreibt alle Writeltems des angegebenen WriteBlocks durch den internen Aufruf von AsOpcUac.UaClt_WriteBulk. Die zu schreibenden Werte werden von den gemappten Variablen genommen.

4.2.2.6.3 BrbUaRcCallMethod

```
void BrbUaRcCallMethod(struct BrbUaRcCallMethod* inst)
```

Argumente:

```
struct BrbUaRcCallMethod* inst  
    Zeiger auf die Funktionsblock-Instanz
```

Eingänge:

```
struct BrbUaRunClient_TYP* pRunClient  
    Zeiger auf die RunClient-Struktur (genaue Beschreibung siehe oben)  
UINT nMethodIndex  
    Der 0-basierte Index der Methode laut Reihenfolge im Datenobjekt
```

Ausgänge:

```
BOOL bInit  
    Vor dem Call für einen Zyklus auf 1 zum optionalen Besetzen der Argumente  
UINT nStatus  
    Funktionsblock-Status  
    eBRB_OK  
    eBRB_ERR_NULL_POINTER  
    eBRB_ERR_UA_ERROR  
    eBRB_ERR_UA_NO_ELEMENTS  
    eBRB_ERR_UA_INVALID_INDEX  
    eBRB_ERR_UA_NOT_CONNECTED  
    eBRB_ERR_BUSY  
    1..65534 = Fehlermeldung. Wenn es sich um einen System-Fehler (<50000) handelt, kann die Beschreibung dazu  
    in der Studio-Hilfe gefunden werden.  
UINT nErrorId  
    OpcUa-Fehler-Status  
STRING sErrorId
```


UDINT OpcUa-Fehler-Status als Text
nCallCount
Anzahl der Calls

Beschreibung:

Ruft die angegebene Methode durch den internen Aufruf von `AsOpcUac.UA_MethodCall` auf. Die Werte der Argumente werden von den gemappten Variablen genommen bzw. zurückgeschrieben. Der Ausgang `bInit` geht vor dem Call für einen Zyklus auf 1. Das kann dazu benutzt werden, um applikativ die Eingangs-Argumente zu besetzen.

4.2.2.7 Zugriff auf die internen Daten

Auf alle intern im allokierten Speicher gehaltenen Daten (z.B. vom Datenobjekt eingelesene Parameter oder ermittelte Handles) kann der Anwender über eigene Funktionen mindestens lesend zugreifen (bei einigen Ausnahmen auch schreibend). Diese Daten (z.B. Handle eines `MonitoredItems`) können dann applikativ verwendet werden.

Die lesenden Funktionen (`BrbUaRcGetXXX`) arbeiten mit reinem Speicherzugriff und sind daher sehr performant.

Ist beim Ermitteln der benötigten Daten (z.B. beim Ermitteln eines `NodeHandles`) ein Fehler aufgetreten, so kann dieser damit eingesehen werden.

Achtung: Die internen Daten werden bei diesen Funktionen immer nur auf den übergebenen Zeiger kopiert. Eine Änderung in dieser Kopie bewirkt also keine Änderung an den internen Daten, also auch keine Parameter-Änderung!

Zur Änderung von Parametern eines Elements gibt es entsprechende Funktionsblöcke (`BrbUaRcSetXXX`). Diese arbeiten intern durch asynchronen Aufruf von System-FB's und brauchen daher u.U. mehrere Zyklen.

4.2.2.7.1 BrbUaRcGetSrvNamespace

```
unsigned short BrbUaRcGetSrvNamespace(struct BrbUaRunClient_TYP* pRunClient, unsigned short  
nDatObjNamespaceIndex, struct BrbUaRcNamespace_TYP* pServerNamespace)
```

Argumente:

`struct BrbUaRunClient_TYP* pRunClient`
Zeiger auf die `RunClient`-Struktur (genaue Beschreibung siehe oben)
`UINT nDatObjNamespaceIndex`
Der 0-basierte Index des Namespaces laut Reihenfolge im Datenobjekt
`struct BrbUaRcNamespace_TYP* pServerNamespace`
Zeiger auf eine Struktur, die die Werte aufnimmt. Kann auch 0 sein

Rückgabe:

`UINT`
Der serverseitige Namespace-Index

Beschreibung:

Gibt den Server-Namespace-Index aufgrund des Namespace-Index aus dem Datenobjekt zurück. Ist der Namespace nicht vorhanden, so wird 0 zurückgegeben.

Optional kann zusätzlich ein Zeiger auf folgende Struktur übergeben werden:

	<code>BrbUaRcNamespace_TYP</code>		Daten eines Namespaces
	<code>sNamespaceUri</code>	<code>STRING[nBRBUA_NAMESPACE_URI_CHAR_MAX]</code>	Namespace-Uri aus dem Datenobjekt
	<code>nNamespaceIndex</code>	<code>UINT</code>	Server-Namespace-Index
	<code>nErrorId</code>	<code>DWORD</code>	OpcUa-Fehler-Status

Ist der Zeiger angegeben, wird diese Struktur befüllt. Hier kann auch erkannt werden, ob und welcher Fehler beim Ermitteln des Server-Namespace-Indexes aufgetreten ist (z.B. bei Angabe einer nicht vorhandenen Uri).

4.2.2.7.2 BrbUaRcGetNodeHandle

```
unsigned short BrbUaRcGetNodeHandle(struct BrbUaRunClient_TYP* pRunClient, unsigned short  
nNodeHandleIndex, struct BrbUaRcNodeHandle_TYP* pNodeHandle)
```

Argumente:

`struct BrbUaRunClient_TYP* pRunClient`

Zeiger auf die RunClient-Struktur (genaue Beschreibung siehe oben)

```
UINT nNodeHandleIndex
```

Der 0-basierte Index des NodeHandles laut Reihenfolge im Datenobjekt

```
struct BrbUaRcNodeHandle_TYP* pNodeHandle
```

Zeiger auf eine Struktur, die die Werte aufnimmt.






Rückgabe:

```
UINT
```

eBRB_OK
eBRB_ERR_NULL_POINTER
eBRB_ERR_UA_NO_ELEMENTS
eBRB_ERR_UA_INVALID_INDEX

Beschreibung:

Gibt die ermittelten Daten zu einem im Datenobjekt angegebenen NodeHandle zurück.
Dazu muss ein Zeiger auf folgende Struktur übergeben werden:

 BrbUaRcNodeHandle_TYP		Daten eines NodeHandles
 nDatObjNamespaceIndex	UINT	Namespace-Index laut Datenobjekt
 NodeId	UAnodeID	NodeId laut Datenobjekt
 nErrorId	DWORD	OpcUa-Fehler-Status beim Ermitteln des NodeHandles
 nNodeHandle	DWORD	Ermittelter NodeHandle

Der Server-Namespace-Index der NodeId und das NodeHandle werden vom Cyclic-FB automatisch ermittelt.

Hier kann auch erkannt werden, ob und welcher Fehler dabei aufgetreten ist (z.B. bei Angabe eines nicht vorhandenen Nodes).

4.2.2.7.3 BrbUaRcGetReadBlock

```
unsigned short BrbUaRcGetReadBlock(struct BrbUaRunClient_TYP* pRunClient, unsigned short nReadBlockIndex, struct BrbUaRcReadBlock_TYP* pReadBlock, struct BrbUaRcReadBlockIntern_TYP* pReadBlockIntern)
```

Argumente:

```
struct BrbUaRunClient_TYP* pRunClient
```

Zeiger auf die RunClient-Struktur (genaue Beschreibung siehe oben)

```
UINT nReadBlockIndex
```

Der 0-basierte Index des ReadBlocks laut Reihenfolge im Datenobjekt

```
struct BrbUaRcReadBlock_TYP* pReadBlock
```

Zeiger auf eine Struktur, die die Werte aufnimmt.

```
struct BrbUaRcReadBlockIntern_TYP* pReadBlockIntern
```

Optionaler Zeiger auf eine Struktur, die auch die internen Werte aufnimmt. Kann auch 0 sein



Rückgabe:

```
UINT
```

eBRB_OK
eBRB_ERR_NULL_POINTER
eBRB_ERR_UA_NO_ELEMENTS
eBRB_ERR_UA_INVALID_INDEX

Beschreibung:

Gibt die ermittelten Daten zu einem im Datenobjekt angegebenen ReadBlock zurück.
Dazu muss ein Zeiger auf folgende Struktur übergeben werden:

 BrbUaRcReadBlock_TYP		Daten eines ReadBlocks
 nReadItemCount	UINT	Anzahl der ReadItems

Optional kann auch ein Zeiger auf eine erweiterte Struktur angegeben werden, welche zusätzliche Informationen beinhaltet (z.B. Zeiger auf den allokierten Speicher). Da diese für den Anwender in der Regel nicht interessant sind, kann hier auch 0 übergeben werden.

4.2.2.7.4 BrbUaRcGetReadItem

```
unsigned short BrbUaRcGetReadItem(struct BrbUaRunClient_TYP* pRunClient, unsigned short nReadBlockIndex, unsigned short nReadItemIndex, struct BrbUaRcReadItem_TYP* pReadItem)
```

Argumente:

```
struct BrbUaRunClient_TYP* pRunClient
```

Zeiger auf die RunClient-Struktur (genaue Beschreibung siehe oben)

```
UINT nReadBlockIndex
    Der 0-basierte Index des ReadBlocks laut Reihenfolge im Datenobjekt
UINT nReadItemIndex
    Der 0-basierte Index des ReadItems laut Reihenfolge im Datenobjekt
struct BrbUaRcReadItem_TYP* pReadItem
    Zeiger auf eine Struktur, die die Werte aufnimmt.
```

Rückgabe:

```
UINT
    eBRB_OK
    eBRB_ERR_NULL_POINTER
    eBRB_ERR_UA_NO_ELEMENTS
    eBRB_ERR_UA_INVALID_INDEX
```

Beschreibung:

Gibt die ermittelten Daten zu einem im Datenobjekt angegebenen ReadItem zurück.
Dazu muss ein Zeiger auf folgende Struktur übergeben werden:

BrbUaRcReadItem_TYP			Daten eines ReadItems
nDatObjNamespaceIndex	UINT		Namespace-Index laut Datenobjekt
NodeId	UANodeID		NodeId laut Datenobjekt
AddInfo	UANodeAdditionalInfo		Zusatzangaben für den Zugriff
sVar	STRING[nBRBUA_VARNAME_TEXT_CHAR_MAX]		Name der gemappten Variable
dtTimestamp	DATE_AND_TIME		Zeitstempel der letzten Aktion
nErrorId	DWORD		OpcUa-Fehler-Status

Hier kann auch erkannt werden, ob und welcher Fehler beim Lesen aufgetreten ist (z.B. bei Angabe eines nicht vorhandenen Nodes).

4.2.2.7.5 BrbUaRcGetWriteBlock

```
unsigned short BrbUaRcGetWriteBlock(struct BrbUaRunClient_TYP* pRunClient, unsigned short
nWriteBlockIndex, struct BrbUaRcWriteBlock_TYP* pWriteBlock, struct BrbUaRcWriteBlockIntern_TYP*
pWriteBlockIntern)
```

Argumente:

```
struct BrbUaRunClient_TYP* pRunClient
    Zeiger auf die RunClient-Struktur (genaue Beschreibung siehe oben)
UINT nWriteBlockIndex
    Der 0-basierte Index des ReadBlocks laut Reihenfolge im Datenobjekt
struct BrbUaRcWriteBlock_TYP* pWriteBlock
    Zeiger auf eine Struktur, die die Werte aufnimmt.
struct BrbUaRcWriteBlockIntern_TYP* pWriteBlockIntern
    Optionaler Zeiger auf eine Struktur, die auch die internen Werte aufnimmt. Kann auch 0 sein
```

Rückgabe:

```
UINT
    eBRB_OK
    eBRB_ERR_NULL_POINTER
    eBRB_ERR_UA_NO_ELEMENTS
    eBRB_ERR_UA_INVALID_INDEX
```

Beschreibung:

Gibt die ermittelten Daten zu einem im Datenobjekt angegebenen WriteBlock zurück.
Dazu muss ein Zeiger auf folgende Struktur übergeben werden:

BrbUaRcWriteBlock_TYP			Daten eines WriteBlocks
nWriteItemCount	UINT		Anzahl der WriteItems

Optional kann auch ein Zeiger auf eine erweiterte Struktur angegeben werden, welche zusätzliche Informationen beinhaltet (z.B. Zeiger auf den allokierten Speicher). Da diese für den Anwender in der Regel nicht interessant sind, kann hier auch 0 übergeben werden.

4.2.2.7.6 BrbUaRcGetWriteItem

```
unsigned short BrbUaRcGetWriteItem(struct BrbUaRunClient_TYP* pRunClient, unsigned short
nWriteBlockIndex, unsigned short nWriteItemIndex, struct BrbUaRcWriteItem_TYP* pWriteItem)
```

Argumente:

```
struct BrbUaRunClient_TYP* pRunClient
```

Zeiger auf die RunClient-Struktur (genaue Beschreibung siehe oben)

`UINT nWriteBlockIndex`
Der 0-basierte Index des WriteBlocks laut Reihenfolge im Datenobjekt

`UINT nWriteItemIndex`
Der 0-basierte Index des Writeltems laut Reihenfolge im Datenobjekt

`struct BrbUaRcWriteItem_TYP* pWriteItem`
Zeiger auf eine Struktur, die die Werte aufnimmt.

Rückgabe:

`UINT`

`eBRB_OK`
`eBRB_ERR_NULL_POINTER`
`eBRB_ERR_UA_NO_ELEMENTS`
`eBRB_ERR_UA_INVALID_INDEX`

Beschreibung:

Gibt die ermittelten Daten zu einem im Datenobjekt angegebenen Writeltem zurück.
Dazu muss ein Zeiger auf folgende Struktur übergeben werden:

BrbUaRcWriteItem_TYP		Daten eines Writeltems
nDatObjNamespaceIndex	UINT	Namespace-Index laut Datenobjekt
NodeId	UANodeID	NodeId laut Datenobjekt
AddInfo	UANodeAdditionalInfo	Zusatzangaben für den Zugriff
sVar	STRING[nBRBUA_VARNAME_TEXT_CHAR_MAX]	Name der gemappten Variable
nErrorId	DWORD	OpcUa-Fehler-Status

Hier kann auch erkannt werden, ob und welcher Fehler beim Schreiben aufgetreten ist (z.B. bei Angabe eines nicht vorhandenen Nodes).

4.2.2.7.7 BrbUaRcGetMethod

`unsigned short BrbUaRcGetMethod(struct BrbUaRunClient_TYP* pRunClient, unsigned short nMethodIndex, struct BrbUaRcMethod_TYP* pMethod, struct BrbUaRcMethodIntern_TYP* pMethodIntern)`

Argumente:

`struct BrbUaRunClient_TYP* pRunClient`
Zeiger auf die RunClient-Struktur (genaue Beschreibung siehe oben)

`UINT nMethodIndex`
Der 0-basierte Index der Method laut Reihenfolge im Datenobjekt

`struct BrbUaRcMethod_TYP* pMethod`
Zeiger auf eine Struktur, die die Werte aufnimmt.

`struct BrbUaRcMethodIntern_TYP* pMethodIntern`
Optionaler Zeiger auf eine Struktur, die auch die internen Werte aufnimmt. Kann auch 0 sein

Rückgabe:

`UINT`

`eBRB_OK`
`eBRB_ERR_NULL_POINTER`
`eBRB_ERR_UA_NO_ELEMENTS`
`eBRB_ERR_UA_INVALID_INDEX`

Beschreibung:

Gibt die ermittelten Daten zu einer im Datenobjekt angegebenen Method zurück.
Dazu muss ein Zeiger auf folgende Struktur übergeben werden:

BrbUaRcMethod_TYP		Daten einer Method
nDatObjNamespaceIndex	UINT	Namespace-Index laut Datenobjekt
ObjectNodeId	UANodeID	NodeId laut Datenobjekt
MethodNodeId	UANodeID	NodeId laut Datenobjekt
tTimeout	TIME	Timeout für den Call
nInputArgsCount	USINT	Anzahl der Eingangs-Argumente
nOutputArgsCount	USINT	Anzahl der Ausgangs-Argumente
nErrorId	DWORD	OpcUa-Fehler-Status beim Ermitteln des MethodHandles
nMethodHandle	DWORD	Ermittelter MethodHandle

Werden im Datenobjekt keine Argumente angegeben, wird trotzdem der MethodHandle ermittelt.
Damit kann dann ein applikativer Call implementiert werden.
Der Server-Namespace-Index der NodeId's und das NodeHandle werden vom Cyclic-FB automatisch ermittelt.

Hier kann auch erkannt werden, ob und welcher Fehler aufgetreten ist (z.B. bei Angabe eines nicht vorhandenen Nodes).

Optional kann auch ein Zeiger auf eine erweiterte Struktur angegeben werden, welche zusätzliche Informationen beinhaltet (z.B. Zeiger auf den allokierten Speicher). Da diese für den Anwender in der Regel nicht interessant sind, kann hier auch 0 übergeben werden.

4.2.2.7.8 BrbUaRcGetArgument

```
unsigned short BrbUaRcGetArgument(struct BrbUaRunClient_TYP* pRunClient, unsigned short nMethodIndex, plcbit bOutput, unsigned short nArgumentIndex, struct UAMethodArgument* pArgument)
```

Argumente:

```
struct BrbUaRunClient_TYP* pRunClient  
    Zeiger auf die RunClient-Struktur (genaue Beschreibung siehe oben)  
UINT nMethodIndex  
    Der 0-basierte Index der Method laut Reihenfolge im Datenobjekt  
BOOL bOutput  
    0=Eingangs-Argument, 1=Ausgangs-Argument  
UINT nArgumentIndex  
    Der 0-basierte Index des Arguments laut Reihenfolge im Datenobjekt  
struct UAMethodArgument_TYP* pArgument  
    Zeiger auf eine Struktur, die die Werte aufnimmt.
```

Rückgabe:

```
UINT  
    eBRB_OK  
    eBRB_ERR_NULL_POINTER  
    eBRB_ERR_UA_NO_ELEMENTS  
    eBRB_ERR_UA_INVALID_INDEX
```

Beschreibung:

Gibt die ermittelten Daten zu einem im Datenobjekt angegebenen Argument zurück.

Dazu muss ein Zeiger auf folgende Struktur aus der System-Bibliothek AsOpcUac übergeben werden:

UAMethodArgument		
Name	STRING[64]	Method argument used to supply the input and receive the output argument values for method calls
Value	STRING[255]	Name of the method argument Source/Destination variable name for the argument value

4.2.2.7.9 BrbUaRcGetSubscription

```
unsigned short BrbUaRcGetSubscription(struct BrbUaRunClient_TYP* pRunClient, unsigned short nSubscriptionIndex, struct BrbUaRcSubscription_TYP* pSubscription, struct BrbUaRcSubscriptionIntern_TYP* pSubscriptionIntern)
```

Argumente:

```
struct BrbUaRunClient_TYP* pRunClient  
    Zeiger auf die RunClient-Struktur (genaue Beschreibung siehe oben)  
UINT nSubscriptionIndex  
    Der 0-basierte Index der Subscription laut Reihenfolge im Datenobjekt  
struct BrbUaRcSubscription_TYP* pSubscription  
    Zeiger auf eine Struktur, die die Werte aufnimmt.  
struct BrbUaRcSubscriptionIntern_TYP* pSubscriptionIntern  
    Optionaler Zeiger auf eine Struktur, die auch die internen Werte aufnimmt. Kann auch 0 sein
```

Rückgabe:

```
UINT  
    eBRB_OK  
    eBRB_ERR_NULL_POINTER  
    eBRB_ERR_UA_NO_ELEMENTS  
    eBRB_ERR_UA_INVALID_INDEX
```

Beschreibung:

Gibt die ermittelten Daten zu einer im Datenobjekt angegebenen Subscription zurück.

Dazu muss ein Zeiger auf folgende Struktur übergeben werden:

BrbUaRcSubscription_TYP		Daten einer Subscription
bPublishingEnable	BOOL	Aktivierung der Subscription
nPriority	USINT	Priorität (0..255, je größer desto höher)
tPublishingInterval	TIME	Sende-Intervall in [ms]
nMonitoredItemCount	UINT	Anzahl der MonitoredItems
nEventItemCount	UINT	Anzahl der EventItems
nErrorId	DWORD	OpcUa-Fehler-Status
nSubscriptionHandle	DWORD	Ermitteltes SubscriptionHandle

Hier kann auch erkannt werden, ob und welcher Fehler beim Erzeugen der Subscription aufgetreten ist.

Optional kann auch ein Zeiger auf eine erweiterte Struktur angegeben werden, welche zusätzliche Informationen beinhaltet (z.B. Zeiger auf den allokierten Speicher). Da diese für den Anwender in der Regel nicht interessant sind, kann hier auch 0 übergeben werden.

4.2.2.7.10 BrbUaRcSetSubscription

```
void BrbUaRcSetSubscription(struct BrbUaRcSetSubscription* inst)
```

Argumente:

```
struct BrbUaRcSetSubscription* inst  
    Zeiger auf die Funktionsblock-Instanz
```

Eingänge:

```
struct BrbUaRunClient_TYP* pRunClient  
    Zeiger auf die RunClient-Struktur (genaue Beschreibung siehe oben)  
UINT nSubscriptionIndex  
    Der 0-basierte Index der Subscription laut Reihenfolge im Datenobjekt  
struct BrbUaRcSubscription_TYP* pSubscription  
    Zeiger auf eine Struktur, die die zu ändernden Werte enthält
```

Ausgänge:

```
UINT nStatus  
    Funktionsblock-Status  
    eBRB_OK  
    eBRB_ERR_NULL_POINTER  
    eBRB_ERR_UA_ERROR  
    eBRB_ERR_UA_NO_ELEMENTS  
    eBRB_ERR_UA_INVALID_INDEX  
    eBRB_ERR_UA_NOT_CONNECTED  
    eBRB_ERR_BUSY  
    1..65534 = Fehlermeldung. Wenn es sich um einen System-Fehler (<50000) handelt, kann die Beschreibung dazu  
    in der Studio-Hilfe gefunden werden.  
BOOL bPublished  
    Daten veröffentlicht seit dem letzten Aufruf  
UINT nErrorId  
    OpcUa-Fehler-Status  
STRING sErrorId  
    OpcUa-Fehler-Status als Text
```

Beschreibung:

Ändert die Parameter einer Subscription laut der übergebenen Werte. Dazu muss dieselbe Struktur wie beim `BrbUaRcGetSubscription` übergeben werden:

BrbUaRcSubscription_TYP		Daten einer Subscription
bPublishingEnable	BOOL	Aktivierung der Subscription
nPriority	USINT	Priorität (0..255, je größer desto höher)
tPublishingInterval	TIME	Sende-Intervall in [ms]
nMonitoredItemCount	UINT	Anzahl der MonitoredItems
nEventItemCount	UINT	Anzahl der EventItems
nErrorId	DWORD	OpcUa-Fehler-Status
nSubscriptionHandle	DWORD	Ermitteltes SubscriptionHandle

Folgende Parameter können damit geändert werden:

```
bPublishingEnable  
nPriority
```

tPublishingInterval

Alle anderen Elemente der Struktur werden nicht berücksichtigt.

Am einfachsten ist es, die aktuellen Werte mit `Get` zu holen, den oder die entsprechenden Parameter abzuändern und dann `Set` aufzurufen.

War der Aufruf erfolgreich, werden die geänderten Parameter auch in die internen Daten übernommen.

Da der intern aufgerufene FB `UA_SubscriptionOperate` auch den Ausgang `Published` besitzt (siehe AS-Hilfe), wird dieser an den Ausgang `bPublished` übernommen.

4.2.2.7.11 BrbUaRcGetMonitoredItem

```
unsigned short BrbUaRcGetMonitoredItem(struct BrbUaRunClient_TYP* pRunClient, unsigned short
nSubscriptionIndex, unsigned short nMonitoredItemIndex, struct BrbUaRcMonitoredItem_TYP* pMoni-
toredItem)
```

Argumente:

`struct BrbUaRunClient_TYP* pRunClient`
Zeiger auf die `RunClient`-Struktur (genaue Beschreibung siehe oben)

`UINT nSubscriptionIndex`
Der 0-basierte Index der Subscription laut Reihenfolge im Datenobjekt

`UINT nMonitoredItemIndex`
Der 0-basierte Index des MonitoredItems laut Reihenfolge im Datenobjekt

`struct BrbUaRcMonitoredItem_TYP* pMonitoredItem`
Zeiger auf eine Struktur, die die Werte aufnimmt.

Rückgabe:

`UINT`

`eBRB_OK`
`eBRB_ERR_NULL_POINTER`
`eBRB_ERR_UA_NO_ELEMENTS`
`eBRB_ERR_UA_INVALID_INDEX`

Beschreibung:

Gibt die ermittelten Daten zu einem im Datenobjekt angegebenen `MonitoredItem` zurück.

Dazu muss ein Zeiger auf folgende Struktur übergeben werden:

BrbUaRcMonitoredItem_TYP			Daten eines MonitoredItems
nDatObjNamespaceIndex	UINT		Namespace-Index laut Datenobjekt
NodeId	UANodeID		NodeId laut Datenobjekt
nNodeHandleErrorId	DWORD		OpcUa-Fehler-Status
nNodeHandle	DWORD		Ermitteltes NodeHandle
AddInfo	UANodeAdditionalInfo		Zusatzangaben für den Zugriff
sVar	STRING[nBRBUA_VARNAME_TEXT_CHAR_MAX]		Name der gemappten Variable
nQueueSizeOri	UINT		Puffergröße laut Datenobjekt
MonitorSettings	UAMonitoringParameters		Überwachungs-Einstellungen
bValueChanged	BOOL		1=Wert hat sich geändert
nRemainingValueCount	UINT		Anzahl der verbleibenden Wert-Änderungen
dtTimestamp	DATE_AND_TIME		Zeitstempel der letzten Wert-Änderung
nNodeQualityId	DWORD		OpcUa-Fehler-Status der letzten Wert-Änderung
nMonitoredItemErrorId	DWORD		OpcUa-Fehler-Status
nMonitoredItemHandle	DWORD		Ermitteltes MonitoredItemHandle

Der Server-Namespaces-Index der `NodeId`, das `NodeHandle` und das `MonitoredItemHandle` werden vom `Cyclic-FB` automatisch ermittelt.

Hier kann auch erkannt werden, ob und welche Fehler aufgetreten sind.

`nQueueSizeOri`

Beim Anlegen des `MonitoredItems` kann der Server bestimmte Einstellungen korrigieren, u.a. auch die `QueueSize`. So wird eine `QueueSize` von 0 serverseitig auf immer 1 korrigiert. Clientseitig aber hat eine `QueueSize` von 0 eine andere Bedeutung als >0:

0 = Firmware-Sync-Modus: Die Werte werden beim Empfang automatisch auf die gemappten Variablen geschrieben

>0 = Controller-Sync-Modus: Die Werte werden nicht automatisch auf die gemappten Variablen geschrieben, sondern erst beim Aufruf von `UA_MonitoredItemOperate` (wird vom Cyclic-FB **nicht** automatisch aufgerufen, sondern muss applikativ gemacht werden).

Dieser Unterschied ist in der AS-Hilfe zum Baustein `UA_MonitoredItemAddList` beschrieben. Wenn nun die RunClient-Verbindung über Kommandos getrennt und anschließend wieder aufgebaut wird und dabei die korrigierte QueueSize verwendet werden würde, würde das eine Änderung des Verhaltens bedeuten. Damit dies nicht passiert, wird die original im Datenobjekt angegebene QueueSize extra gespeichert und bei jedem Verbindungs-Aufbau wiederverwendet. Die korrigierte QueueSize ist Bestandteil vom Element `MonitorSettings`.

Einige der anderen Parameter von `MonitorSettings` können auch vom Server korrigiert werden (z.B. `SamplingInterval`) und werden dann hier entsprechend angezeigt. Da eine Korrektur dieser Parameter bei einer Wieder-Anmeldung aber kein Problem darstellt, werden diese Daten nicht doppelt gehalten.

`bValueChanged`

Dieses Flag wird bei jeder empfangenen Wert-Änderung vom Betriebssystem auf 1 gesetzt, aber nicht gelöscht. Sehr einfach ausgelesen und optional auch gelöscht werden kann es durch die Funktion `BrbUaRcGetMiValueChanged` (siehe unten). Damit kann applikativ gezielt jede Wert-Änderungen erkannt und darauf reagiert werden.

`nRemainingValueCount`

Bei einer QueueSize > 0 werden alle Wert-Änderungen gesendet, die seit dem letzten PublishingInterval erkannt wurden, also der Wert-Änderungs-Puffer. Dieser muss mit `UA_MonitoredItemOperate` Wert für Wert ausgelesen werden (wird vom Cyclic-FB **nicht** erledigt). `nRemainingValueCount` gibt nun an, wieviel Eintragungen in diesen Puffer noch nicht abgeholt wurden. Zum vereinfachten Ermitteln dieses Zählers kann auch die Funktion `BrbUaRcGetMiRemainingValueCount` verwendet werden (siehe unten).

`dtTimestamp`

Enthält den Zeitstempel der letzten Wert-Änderung.

`nNodeQuality`

Enthält den Qualitäts-Status der letzten Wert-Änderung als OpcUa-Status.

Hinweis: Nähere Beschreibungen zu den einzelnen Werten siehe AS-Hilfe zum FB `UA_MonitoredItemAddList`.

4.2.2.7.12 BrbUaRcSetMonitoredItem

```
void BrbUaRcSetMonitoredItem(struct BrbUaRcSetMonitoredItem* inst)
```

Argumente:

```
struct BrbUaRcSetMonitoredItem* inst  
    Zeiger auf die Funktionsblock-Instanz
```

Eingänge:

```
struct BrbUaRunClient_TYP* pRunClient  
    Zeiger auf die RunClient-Struktur (genaue Beschreibung siehe oben)  
UINT nSubscriptionIndex  
    Der 0-basierte Index der Subscription laut Reihenfolge im Datenobjekt  
    UINT nMonitoredItemIndex  
    Der 0-basierte Index des MonitoredItems laut Reihenfolge im Datenobjekt  
struct UAMonitoringSettings_TYP* pMonitoringSettings  
    Zeiger auf eine Struktur, die die zu ändernden Werte enthält
```

Ausgänge:

```
UINT nStatus  
    Funktionsblock-Status  
    eBRB_OK
```



```
eBRB_ERR_NULL_POINTER
eBRB_ERR_UA_ERROR
eBRB_ERR_UA_NO_ELEMENTS
eBRB_ERR_UA_INVALID_INDEX
eBRB_ERR_UA_NOT_CONNECTED
eBRB_ERR_BUSY
1..65534 = Fehlermeldung. Wenn es sich um einen System-Fehler (<50000) handelt, kann die Beschreibung dazu
in der Studio-Hilfe gefunden werden.
BOOL bValueChanged
    Der Wert der überwachten Variable hat sich geändert
DATE_AND_TIME dtTimestamp
    Zeitstempel der Änderung
UINT nErrorId
    OpcUa-Fehler-Status
STRING sErrorId
    OpcUa-Fehler-Status als Text
```

Beschreibung:

Ändert die Parameter eines MonitoredItems laut der übergebenen Werte. Dazu muss folgende Struktur aus der System-Bibliothek AsOpcUac übergeben werden:

<div> <div>UAMonitoringSettings</div> <div> <div>SamplingInterval</div> <div>DeadbandType</div> <div>Deadband</div> </div> </div>		<div> TIME UADeabandType REAL </div>	<div> Settings for data item monitoring Time in ms Deadband type which applies to deadband value Deadband value, semantics depending on DeadbandType </div>
Enum-Konstantee	Wert	Beschreibung	
UADeabandType_None	0	Es sollte keine Deadband-Berechnung angewendet werden.	
UADeabandType_Absolute	1	Der angegebene Deadband-Wert wird als absoluter Wert interpretiert. Um eine Wertaufzeichnung auszulösen, muss der Betrag der Differenz des aktuellen Werts und des letzten Samples über diesem Wert liegen.	
UADeabandType_Percentt	2	Der angegebene Deadband-Wert wird als prozentueller Wert vom EURange interpretiert.	

War der Aufruf erfolgreich, werden die geänderten Parameter auch in die internen Daten übernommen.

Da der intern aufgerufene FB UA_MonitoredItemOperate auch die Ausgänge ValueChanged und TimeStamp besitzt (siehe AS-Hilfe), werden diese an den Ausgängen bValueChanged und dtTimestamp übernommen.

4.2.2.7.13 BrbUaRcGetMiValueChanged

```
plcbit BrbUaRcGetMiValueChanged(struct BrbUaRunClient_TYP* pRunClient, unsigned short nSubscriptionIndex, unsigned short nMonitoredItemIndex, plcbit bClear)
```

Argumente:

```
struct BrbUaRunClient_TYP* pRunClient
    Zeiger auf die RunClient-Struktur (genaue Beschreibung siehe oben)
UINT nSubscriptionIndex
    Der 0-basierte Index der Subscription laut Reihenfolge im Datenobjekt
UINT nMonitoredItemIndex
    Der 0-basierte Index des MonitoredItems laut Reihenfolge im Datenobjekt
BOOL bClear
    1 = Flag wird gelöscht
```

Rückgabe:

```
BOOL
    0 = Flag nicht gesetzt
    1 = Flag gesetzt
```

Beschreibung:

Dieses Flag wird bei jeder empfangenen Wert-Änderung vom Betriebssystem auf 1 gesetzt, aber nicht mehr gelöscht.

Mit dieser Funktion kann es sehr einfach ausgelesen und optional auch gelöscht werden. Damit kann applikativ gezielt jede Wert-Änderungen erkannt und darauf reagiert werden.

4.2.2.7.14 BrbUaRcGetMiRemainingValueCount

```
unsigned short BrbUaRcGetMiRemainingValueCount(struct BrbUaRunClient_TYP* pRunClient, unsigned short nSubscriptionIndex, unsigned short nMonitoredItemIndex)
```

Argumente:

```
struct BrbUaRunClient_TYP* pRunClient  
    Zeiger auf die RunClient-Struktur (genaue Beschreibung siehe oben)  
UINT nSubscriptionIndex  
    Der 0-basierte Index der Subscription laut Reihenfolge im Datenobjekt  
UINT nMonitoredItemIndex  
    Der 0-basierte Index des MonitoredItems laut Reihenfolge im Datenobjekt
```

Rückgabe:

```
BOOL  
    Zähler der verbliebenen Wert-Änderungen
```

Beschreibung:

Bei einer QueueSize > 0 werden alle Wert-Änderungen gesendet, die seit dem letzten PublishingInterval erkannt wurden, also der Wert-Änderungs-Puffer. Die Werte werden nicht automatisch auf die gemappte Variable kopiert. Vielmehr muss der Puffer mit UA_MonitoredItemOperate Wert für Wert ausgelesen werden. Dies wird vom Cyclic-FB **nicht** gemacht, sondern muss applikativ erledigt werden. Der Rückgabe-Wert dieser Funktion gibt nun an, wieviel Eintragungen in diesen Puffer noch nicht abgeholt wurden.

4.2.2.7.15 BrbUaRcGetEventItem

```
unsigned short BrbUaRcGetEventItem(struct BrbUaRunClient_TYP* pRunClient, unsigned short nSubscriptionIndex, unsigned short nEventItemIndex, struct BrbUaRcEventItem_TYP* pEventItem, struct BrbUaRcEventItemIntern_TYP* pEventItemIntern)
```

Argumente:

```
struct BrbUaRunClient_TYP* pRunClient  
    Zeiger auf die RunClient-Struktur (genaue Beschreibung siehe oben)  
UINT nSubscriptionIndex  
    Der 0-basierte Index der Subscription laut Reihenfolge im Datenobjekt  
UINT nEventItemIndex  
    Der 0-basierte Index des EventItems laut Reihenfolge im Datenobjekt  
struct BrbUaRcEventItem_TYP* pEventItem  
    Zeiger auf eine Struktur, die die Werte aufnimmt.  
struct BrbUaRcEventItemIntern_TYP* pEventItemIntern  
    Optionaler Zeiger auf eine Struktur, die auch die internen Werte aufnimmt. Kann auch 0 sein
```

Rückgabe:

```
UINT  
    eBRB_OK  
    eBRB_ERR_NULL_POINTER  
    eBRB_ERR_UA_NO_ELEMENTS  
    eBRB_ERR_UA_INVALID_INDEX
```

Beschreibung:

Gibt die ermittelten Daten zu einem im Datenobjekt angegebenen EventItem zurück.

Dazu muss ein Zeiger auf folgende Struktur übergeben werden:

BrbUaRcEventItem_TYP			Daten eines EventItems
nEventDatObjNamespaceIndex	UINT		Namespace-Index laut Datenobjekt
EventNodeId	UANodeId		NodeId laut Datenobjekt
nEventNodeHandleErrorId	DWORD		OpcUa-Fehler-Status
nEventNodeHandle	DWORD		Ermitteltes NodeHandle
nTypeDatObjNamespaceIndex	UINT		Namespace-Index laut Datenobjekt
TypeNodeId	UANodeId		NodeId laut Datenobjekt
nEventItemErrorId	DWORD		OpcUa-Fehler-Status
nEventItemHandle	DWORD		Ermitteltes NodeHandle
tTimeout	TIME		Timeout für Operate in [ms]
bCallOperate	BOOL		1=Zyklischer Aufruf von Operate
nEventFieldCount	UINT		Anzahl der Fields
nReceiveCount	UDINT		Empfangs-Zähler
bEventReceived	BOOL		1=Event wurde empfangen (nur für einen Zyklus)

Der Server-Namespaces-Index der NodeId's, die NodeHandles und das EventItemHandle werden vom Cyclic-FB automatisch ermittelt.

Hier kann auch erkannt werden, ob und welche Fehler aufgetreten sind.

Optional kann auch ein Zeiger auf eine erweiterte Struktur angegeben werden, welche zusätzliche Informationen beinhaltet (z.B. Zeiger auf den allokierten Speicher). Da diese für den Anwender in der Regel nicht interessant sind, kann hier auch 0 übergeben werden.

bCallOperate

Die Field-Werte von empfangenen Events werden vom Betriebssystem **nicht** automatisch auf die gemappten Variablen kopiert. Vielmehr geschieht dies erst mit dem Aufruf von UA_EventItemOperate, damit der Anwender auf den Empfang reagieren kann.

Mit diesem Parameter wurde im Datenobjekt festgelegt, ob der Cyclic-FB diesen Aufruf machen soll. Zur Empfangs-Erkennung gibt es dann die Funktion BrbUaRcGetEventItemReceived (siehe unten).

nReceiveCount

Dieser Zähler läuft mit, wenn bCallOperate auf 1 ist. Er gibt die Anzahl der empfangenen Events an. Zum vereinfachten Ermitteln dieses Zählers kann auch die Funktion BrbUaRcGetEventItemReceiveCount verwendet werden (siehe unten).

bEventReceived

Wenn bCallOperate auf 1 ist, steht dieses Flag für einen Zyklus an, wenn ein Event empfangen wurde. So kann der Anwender applikativ darauf reagieren.

Zum vereinfachten Auslesen dieses Flags kann auch die Funktion BrbUaRcGetEventItemReceived verwendet werden (siehe unten).

Hinweis: Nähere Beschreibungen zu den einzelnen Werten siehe AS-Hilfe zum FB UA_EventItemAdd und UA_EventItemOperate.

4.2.2.7.16 BrbUaRcGetEventItemReceiveCount

```
unsigned long BrbUaRcGetEventItemReceiveCount(struct BrbUaRunClient_TYP* pRunClient, unsigned short nSubscriptionIndex, unsigned short nEventItemIndex)
```

Argumente:

```
struct BrbUaRunClient_TYP* pRunClient  
    Zeiger auf die RunClient-Struktur (genaue Beschreibung siehe oben)  
UINT nSubscriptionIndex  
    Der 0-basierte Index der Subscription laut Reihenfolge im Datenobjekt  
UINT nEventItemIndex  
    Der 0-basierte Index des EventItems laut Reihenfolge im Datenobjekt
```

Rückgabe:

```
UDINT  
    Empfangszähler
```

Beschreibung:

Wird diese Funktion zyklisch nach dem Cyclic-FB aufgerufen, kann damit sehr einfach ermittelt werden, wie oft ein Event seit dem Connect empfangen wurde.

Voraussetzung dafür ist, dass der Parameter CallOperate für dieses EventItem im Datenobjekt auf 1 gesetzt wurde.

4.2.2.7.17 BrbUaRcGetEventItemReceived

```
plcbit BrbUaRcGetEventItemReceived(struct BrbUaRunClient_TYP* pRunClient, unsigned short nSubscriptionIndex, unsigned short nEventItemIndex)
```

Argumente:

```
struct BrbUaRunClient_TYP* pRunClient  
    Zeiger auf die RunClient-Struktur (genaue Beschreibung siehe oben)  
UINT nSubscriptionIndex  
    Der 0-basierte Index der Subscription laut Reihenfolge im Datenobjekt
```

UINT nEventItemIndex

Der 0-basierte Index des EventItems laut Reihenfolge im Datenobjekt

Rückgabe:

BOOL

0 = Event wurde nicht empfangen

1 = Event wurde empfangen

Beschreibung:

Wird diese Funktion zyklisch nach dem Cyclic-FB aufgerufen, kann damit sehr einfach der Erhalt eines Events detektiert werden. Damit kann applikativ gezielt auf Event-Empfang reagiert werden. Voraussetzung dafür ist, dass der Parameter `CallOperate` für dieses EventItem im Datenobjekt auf 1 gesetzt wurde.

4.2.2.7.18 BrbUaRcGetEventField

```
unsigned short BrbUaRcGetEventField(struct BrbUaRunClient_TYP* pRunClient, unsigned short nSubscriptionIndex, unsigned short nEventItemIndex, unsigned short nEventFieldIndex, struct BrbUaRcEventField_TYP* pEventField)
```

Argumente:

struct BrbUaRunClient_TYP* pRunClient

Zeiger auf die RunClient-Struktur (genaue Beschreibung siehe oben)

UINT nSubscriptionIndex

Der 0-basierte Index der Subscription laut Reihenfolge im Datenobjekt

UINT nEventItemIndex

Der 0-basierte Index des EventItems laut Reihenfolge im Datenobjekt

UINT nEventFieldIndex

Der 0-basierte Index des EventFields laut Reihenfolge im Datenobjekt

struct BrbUaRcEventField_TYP* pEventField

Zeiger auf eine Struktur, die die Werte aufnimmt.

Rückgabe:

UINT

eBRB_OK

eBRB_ERR_NULL_POINTER

eBRB_ERR_UA_NO_ELEMENTS

eBRB_ERR_UA_INVALID_INDEX

Beschreibung:

Gibt die ermittelten Daten zu einem im Datenobjekt angegebenen EventField zurück.

Dazu muss ein Zeiger auf folgende Struktur übergeben werden:

BrbUaRcEventField_TYP			Daten eines EventFields
nDatObjNamespaceIndex	UINT		Namespace-Index laut Datenobjekt
FieldSelection	UARelativePath		Angaben zum Field
sVar	STRING[nBRBUA_VARNAME_TEXT_CHAR_MAX]		Name der gemappten Variable

Die Unterstruktur `FieldSelection` stammt aus der System-Bibliothek `AsOpcUac`:

UARelativePath			Relative path used to translate and browse nodes
NoOfElements	DWORD		
Elements	UARelativePathElement[0..MAX_INDEX_RELATIVEPATH]		
UARelativePathElement			Relative path element used to build a relative path
ReferenceTypeld	UANodeID		
IsInverse	BOOL		
IncludeSubTypes	BOOL		
TargetName	UAQualifiedName		

Hier wird allerdings nur das Element `TargetName` verwendet:

UAQualifiedName			Qualified name type
NamespaceIndex	UINT		
Name	STRING[255]		

Hinweis: Nähere Beschreibungen zu den einzelnen Werten siehe AS-Hilfe zum FB

`UA_EventItemAdd`.

4.2.2.8 BrbUaRcMonitor

```
unsigned short BrbUaRcMonitor(struct BrbUaRunClient_TYP* pRunClient, struct BrbUaRcMonitor_TYP* pMonitor)
```

Argumente:

```
struct BrbUaRunClient_TYP* pRunClient  
    Zeiger auf die RunClient-Struktur (genaue Beschreibung siehe oben)  
struct BrbUaRcMonitor_TYP* pMonitor  
    Zeiger auf die Monitor-Struktur
```

Rückgabe:

```
UINT  
    eBRB_OK  
    eBRB_ERR_NULL_POINTER  
    eBRB_ERR_NOT_ENABLED
```

Beschreibung:

Mit dieser Funktion kann sehr einfach ein Monitor zum Anzeigen der internen Daten des RunClients implementiert werden. Da hier auch evtl. aufgetretene Fehler-Codes ausgelesen werden können, wird dadurch die Entwicklung und Inbetriebnahme sehr erleichtert.

Dazu muss ein Zeiger auf folgende Struktur übergeben werden:

BrbUaRcMonitor_TYP			Monitor für interne Daten
bEnable	BOOL		1=Aktiv
nMonitorStatus	UINT		Status des Monitors
Namespace	BrbUaRcMonitorNamespace_TYP		Interne Daten zu einem Namespace
NodeHandle	BrbUaRcMonitorNodeHandle_TYP		Interne Daten zu einem NodeHandle
ReadBlock	BrbUaRcMonitorReadBlock_TYP		Interne Daten zu einem ReadBlock
ReadItem	BrbUaRcMonitorReadItem_TYP		Interne Daten zu einem ReadItem
WriteBlock	BrbUaRcMonitorWriteBlock_TYP		Interne Daten zu einem WriteBlock
WriteItem	BrbUaRcMonitorWriteItem_TYP		Interne Daten zu einem WriteItem
Method	BrbUaRcMonitorMethod_TYP		Interne Daten zu einer Method
Argument	BrbUaRcMonitorArgument_TYP		Interne Daten zu einem Argument
Subscription	BrbUaRcMonitorSubscription_TYP		Interne Daten zu einer Subscription
MonitoredItem	BrbUaRcMonitorMonitoredItem_TYP		Interne Daten zu einem MonitoredItem
EventItem	BrbUaRcMonitorEventItem_TYP		Interne Daten zu einem EventItem
EventField	BrbUaRcMonitorEventField_TYP		Interne Daten zu einem EventField

Durch `bEnable` wird der Monitor aktiviert.

Gibt es ein grundsätzliches Problem, wird dies in `nMonitorStatus` (siehe auch Rückgabewert der Funktion) angegeben.

Jede dieser Unterstrukturen bietet mithilfe der oben beschriebenen Zugriffs-Funktionen die Anzeige der internen Daten eines bestimmten Elements, welches im Datenobjekt angegeben wurde (z.B. eines MonitoredItems).

Bei jeder Unterstruktur kann der Index des Elements (bzw. mehrere Indizes bei Schachtelungen) angegeben werden. Die internen Werte des adressierten Elements werden dann auf die ebenfalls enthaltene Struktur kopiert. Ist dies nicht möglich (z.B. weil ein ungültiger Index angegeben wurde), wird der in der Unterstruktur befindliche `nMonitorStatus` gesetzt.

Die Beschreibung zu den Werten eines Elements siehe oben bei der entsprechenden Get-Funktion.

4.2.2.9 Tipps zur Implementierung, Inbetriebnahme und Fehlersuche

Es wird immer nur der als letztes erkannte Fehler im Status der RunClient-Struktur angezeigt. Es sollte also ein Fehler nach dem anderen behoben werden, bis kein Fehler mehr erkannt wird. Hier folgen einige Tipps zur Diagnose, um Parametrier-Fehler zu erkennen.

Bestimmte Fehler (z.B. beim Ermitteln eines Handles) treten erst auf, wenn die Verbindung mit dem Server aufgebaut werden konnte.

Die einzelnen Einträge sollten im Datenobjekt Schritt für Schritt von oben nach unten eingetragen oder einkommentiert und dann am Zielsystem getestet werden.

Wird im Status der RunClient-Struktur ein Fehler angezeigt, so liefert meist der Fehlertext einige Detail-Infos, z.B. bei welcher Aktion und bei welchem Item der Fehler aufgetreten ist.

Zur genaueren Fehler-Diagnose sollte dann mit dem Monitor dieses Item untersucht werden. Hier finden sich alle ErrorId's, die sich beim Behandeln dieses Items ergeben haben.

4.2.2.9.1 Beispiel für ein MonitoredItem

Beispiel: Der Wert einer auf ein MonitoredItem gemappten Variable ändert sich nicht, obwohl eigentlich Änderungen empfangen werden müssten.

Zuerst sieht man sich den State der RunClient-Struktur im Watch des Tasks an. Er zeigt zum Beispiel folgendes:

State	BrbUaRcState_TYP		
eClientState	BrbUaRcClientStates_EN		eBRB_RCCLTSTATE_CONNECTED
nClientErrorId	UDINT		16#0000_0000
sClientErrorText	STRING[255]		'Cyclic error on adding MonitoredItems from Subscription#0. MonitoredItems #0 to #3'

Somit steht schon mal fest, dass es Schwierigkeiten beim Hinzufügen der MonitoredItems#0..3 der Subscription#0 gab.

Um dies nun genauer zu diagnostizieren, verwendet man den Monitor ebenfalls im Watch. Zuerst wird der Monitor eingeschaltet:

RcMonitor	BrbUaRcMonitor_TYP	local	
bEnable	BOOL		TRUE

Dann wirft man einen Blick auf die MonitoredItems#0..3 der Subscription#0. In diesem Beispiel kann man bei einem der MonitoredItems (nämlich #1) feststellen, dass das MonitoredItemHandle nicht ermittelt werden konnte:

MonitoredItem	BrbUaRcMonitorMonitoredItem		
nSubscriptionIndex	UINT		0
nMonitoredItemIndex	UINT		1
nMonitorStatus	UINT		0
MonitoredItem	BrbUaRcMonitoredItem_T		
nClientNamespaceIndex	UDINT		3
NodeId	UANodeID		
NamespaceIndex	UINT		8
Identifier	STRING[255]		'::ServerData:VarsLocal.ReadOnly.nUsintX'
IdentifierType	UAIdentifierType		UAIdentifierType_String
nNodeHandleErrorId	DWORD		16#8034_0000
nNodeHandle	DWORD		0
AddInfo	UANodeAdditionalInfo		
sVar	STRING[255]		'::BrbUaCltC:ClientVarsSubscription.nUsint'
nQueueSizeOri	UINT		0
MonitorSettings	UAMonitoringParameters		
bValueChanged	BOOL		FALSE
nRemainingValueCount	UINT		0
dtTimestamp	DATE_AND_TIME		DT#1970-01-01-00:00:00
nNodeQualityId	DWORD		0
nMonitoredItemErrorId	DWORD		16#A00C_0000
nMonitoredItemHandle	DWORD		0

Die ErrorId 0xA00C0000 bedeutet laut AS-Hilfe `PlcOpen_BadNodeInvalidHdl`. Dies heißt, dass der NodeHandle des Items nicht ermittelt werden konnte. Tatsächlich kann hier auch festgestellt werden, dass beim Ermitteln des Handles schon einen Fehler auftrat:

MonitoredItem	BrbUaRcMonitorMonitore		
nSubscriptionIndex	UINT		0
nMonitoredItemIndex	UINT		1
nMonitorStatus	UINT		0
MonitoredItem	BrbUaRcMonitoredItem_T		
nClientNamespaceIndex	UDINT		3
NodeId	UANodeID		
NamespaceIndex	UINT		8
Identifier	STRING[255]		'::ServerData:VarsLocal.ReadOnly.nUsintX'
IdentifierType	UAIdentifierType		UAIdentifierType_String
nNodeHandleErrorId	DWORD		16#8034_0000
nNodeHandle	DWORD		0
AddInfo	UANodeAdditionalInfo		
sVar	STRING[255]		'::BrbUaCltC:ClientVarsSubscription.nUsint'
nQueueSizeOri	UINT		0
MonitorSettings	UAMonitoringParameters		
bValueChanged	BOOL		FALSE
nRemainingValueCount	UINT		0
dtTimestamp	DATE_AND_TIME		DT#1970-01-01-00:00:00
nNodeQualityId	DWORD		0
nMonitoredItemErrorId	DWORD		16#A00C_0000
nMonitoredItemHandle	DWORD		0

Die ErrorId 0x80340000 bedeutet laut AS-Hilfe Bad_NodeIdUnknown = Die NodeID bezieht sich auf einen Node, der nicht im Serveradressraum vorhanden ist.

Somit sollte überprüft werden, ob der Node korrekt angegeben ist:

MonitoredItem	BrbUaRcMonitorMonitore		
nSubscriptionIndex	UINT		0
nMonitoredItemIndex	UINT		1
nMonitorStatus	UINT		0
MonitoredItem	BrbUaRcMonitoredItem_T		
nClientNamespaceIndex	UDINT		3
NodeId	UANodeID		
NamespaceIndex	UINT		8
Identifier	STRING[255]		'::ServerData:VarsLocal.ReadOnly.nUsintX'
IdentifierType	UAIdentifierType		UAIdentifierType_String
nNodeHandleErrorId	DWORD		16#8034_0000
nNodeHandle	DWORD		0
AddInfo	UANodeAdditionalInfo		
sVar	STRING[255]		'::BrbUaCltC:ClientVarsSubscription.nUsint'
nQueueSizeOri	UINT		0
MonitorSettings	UAMonitoringParameters		
bValueChanged	BOOL		FALSE
nRemainingValueCount	UINT		0
dtTimestamp	DATE_AND_TIME		DT#1970-01-01-00:00:00
nNodeQualityId	DWORD		0
nMonitoredItemErrorId	DWORD		16#A00C_0000
nMonitoredItemHandle	DWORD		0

Es könnte der NamespaceIndex oder auch der Identifier falsch sein. In diesem Beispiel wurde ein X zu viel an den Knotennamen angehängt.

Jetzt kann der Fehler im Datenobjekt behoben, die Änderung auf das Zielsystem geladen und ein Warmstart ausgeführt werden (das ist wichtig, weil das Datenobjekt nur im Init neu eingelesen wird).

4.2.2.9.2 ReadBlocks, WriteBlocks und Methods

Hier geben die Instanzen der entsprechenden FB's einen Hinweis auf den Fehler.

Z.B. bei 'BrbUaRcReadBlock':

fbBrbUaRcReadBlock0	BrbUaRcReadBlock	local	
pRunClient	UDINT		53203808
nReadBlockIndex	UINT		0
nStatus	UINT		51003
nErrorId	DWORD		16#B004_0000
sErrorId	STRING[96]		'0xB0040000 = PlcOpen_BadVariableNameInvalid'
eStep	DINT		0
fbUaClt_ReadBulk	UaClt_ReadBulk		

Die ErrorId 0xB0040000 bedeutet laut AS-Hilfe PlcOpen_BadVariableNameInvalid = Variablenname ungültig.

Es sollten also die Variablennamen der ReadItems dieses ReadBlocks genauer untersucht werden.
Dazu nutzt man wieder den Monitor:

ReadItem	BrbUaRcMonitorReadItem		
nReadBlockIndex	UINT		0
nReadItemIndex	UINT		1
nMonitorStatus	UINT		0
ReadItem	BrbUaRcReadItem_TYP		
nClientNamespaceIndex	UINT		3
NodeID	UANodeID		
AddInfo	UANodeAdditionalInfo		
sVar	STRING[255]		":BrbUaCltC:ClientVarsRead.nDintX"
dtTimestamp	DATE_AND_TIME		DT#1970-01-01-00:00:00
nErrorId	DWORD		16#B004 0000

In diesem Beispiel wurde am ReadItem#1 ein X zu viel an den Variablennamen angehängt.
Jetzt kann der Fehler im Datenobjekt behoben, die Änderung auf das Zielsystem geladen und ein Warmstart ausgeführt werden (das ist wichtig, weil das Datenobjekt nur im Init neu eingelesen wird).

4.3 Server

In diesem Paket finden sich Datentypen und Funktionen für den Server.

4.3.1 Methods

In diesem Paket finden sich Datentypen und Funktionen zur Behandlung von MethodCalls auf Server-Seite.

4.3.1.1 BrbUaSrvHandleMethod






Um den Aufruf einer Methode applikativ zu behandeln, wird der System-Bibliotheks-FB `UA_Srv_MethodOperate` verwendet.

Die Behandlung bedingt mehrere Schritte (Erkennen des Aufrufs, Ausführen des Codes, Rückmeldung) und kann deshalb am besten mit einer Schrittkette umgesetzt werden.

Die Funktion `BrbUaHandleMethod` kapselt diese Schrittkette und reduziert somit den applikativen Aufwand enorm.

4.3.1.1.1 Struktur

Die zu übergebene Struktur beinhaltet mehrere Unterstrukturen:

		BrbUaSrvMethod_TYP	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Methode
		Par	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Parameter
		State	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Status
		Intern	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Interne Variablen

Die Unterstruktur `Intern` wird nur intern verwendet und darf nicht beschrieben werden.

4.3.1.1.1.1 Par








		BrbUaSrvMethodPar_TYP		<input checked="" type="checkbox"/>	Parameter einer Methode	
		sMethodName	STRING[nBRBUA_METHOD_CHAR_MAX]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Name der Methode
		bEnable	BOOL	<input type="checkbox"/>	<input checked="" type="checkbox"/>	0=Ausführung beenden, 1=Ausführen
		bUserCodeIsFinished	BOOL	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Rückmeldung, dass Benutzer-Code ausgeführt wurde

Bei `sMethodName` muss der Name der Methode angegeben werden, der auch bei der Deklaration in der `*.uam`-Datei verwendet wurde.

Der Eingang `bEnable` muss auf 1 gesetzt werden, damit eine Prüfung des Aufrufs gemacht wird. Beim Setzen auf 0 wird die interne Schrittkette gestoppt (es werden keine weiteren Aufrufe erkannt), ausser es wird gerade der Benutzer-Code ausgeführt. In diesem Fall läuft die Schrittkette solange weiter, bis die Ausführung des Benutzer-Codes zurückgemeldet wurde.

Mit `bUserCodeIsFinished` muss zurückgemeldet werden, dass der für diese Methode geschriebene Benutzer-Code ausgeführt wurde.

4.3.1.1.1.2 State

		BrbUaSrvMethodState_TYP		<input checked="" type="checkbox"/>	Status einer Methode	
		bExecuteUserCode	BOOL	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Meldung, dass Benutzer-Code ausgeführt werden muss
		bExecuteUserCodeInit	BOOL	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Wird nur im ersten Zyklus gesetzt
		nCallCount	UINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Aufruf-Zähler
		nErrorCount	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Fehler-Zähler
		nLastErrorId	UDINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Letzter aufgetretener Fehler
		sLastErrorId	STRING[32]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Letzter aufgetretener Fehler als Hex
		eLastErrorAction	UaMethodOperateAction	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Aktion beim letzten aufgetretenen Fehler

Der Status `bExecuteUserCode` gibt an, dass die Methode aufgerufen wurde und der für diese Methode geschriebene Code ausgeführt werden muss.

Der Status `bExecuteUserCodeInit` wird gleichzeitig mit `bExecuteUserCode` gesetzt, aber nach dem ersten Zyklus wieder gelöscht. So können applikative Initialisierungen pro Aufruf durchgeführt werden.

`nCallCount` und `nErrorCount` sind Zähler und dienen der Diagnose.

Die letzten Stati werden nur im Fehlerfall besetzt.

4.3.1.1.2 BrbUaSrvHandleMethod

```
plcdword BrbUaSrvHandleMethod(struct BrbUaSrvMethod_TYP* pMethod)
```

Argumente:

```
struct BrbUaSrvMethod_TYP* pMethod  
    Zeiger auf die Struktur
```

Rückgabe:

```
DWORD  
    0x00000000 = Good (Kein Fehler)  
    >0 = Aktuell aufgetretener Fehler. Beim nächsten Zyklus wieder 0x00000000
```

Beschreibung:

Diese Funktion behandelt einen Server-Methoden-Aufruf.

Beispiel für den Aufruf:

Initialisierung:

```
memset(&MethodMultiply, 0, sizeof(MethodMultiply));  
strcpy(MethodMultiply.Par.sMethodName, "Multiply");
```

Zyklisch:

```
MethodMultiply.bEnable = 1;  
if(BrbUaSrvHandleMethod(&MethodMultiply) == 0)  
{  
    if(MethodMultiply.State.bExecuteUserCode == 1)  
    {  
        // Benutzer-Code muss ausgeführt werden  
        if(MethodMultiply.State.bExecuteUserCodeInit == 1)  
        {  
            // Optionale Initialisierung pro Aufruf  
            ...  
        }  
  
        // Benutzer-Code  
        VarsMultiply.rC = VarsMultiply.rA * VarsMultiply.rB;  
  
        // Rückmeldung  
        MethodMultiply.Par.bUserCodeIsFinished = 1;  
    }  
    else  
    {  
        // Fehler-Behandlung  
    }  
}
```

Der Benutzer-Code kann auch durchaus mehrere Task-Zyklen brauchen, z.B. kann eine Schrittkette implementiert werden.

Der Init-Teil wird bei jedem Call nur einmalig ausgeführt. So kann z.B. eine nachfolgende Schrittkette initialisiert werden.

4.3.1.2 BrbUaGetMethodOperateActionText

```
plcdword BrbUaSrvGetMethodOperateText(enum UaMethodOperateAction eAction, plcstring* pActionText,  
unsigned long nActionTextSize)
```

Argumente:

```
enum UaMethodOperateAction eAction  
    Angabe der Aktion  
STRING* pActionText  
    Zeiger auf den String, der gefüllt werden soll  
UDINT nActionTextSize  
    Größe des Strings, der gefüllt werden soll
```

Rückgabe:

```
DWORD  
    0x00000000 = Good (Kein Fehler)
```

0x80460000 = Bad_StructureMissing (Nullpointer)

Beschreibung:

Diese Funktion gibt den Text einer OpcUa-Methoden-Aktion zurück, welche als Enumeration in AsOpcUas deklariert ist:

UaMethodOperateAction		
UaMoa_CheckIsCalled	0	Check if a method call is pending
UaMoa_Finished	1	Finish method call

Sie kann zur Diagnose in einer Visu verwendet werden.

4.3.2 RunServer

Diese Sammlung an FB's und Funktionen kapselt Funktionen für den Server. Damit lassen sich ohne großen applikativen Aufwand sonst aufwändige Server-Funktionen implementieren.

Der RunServer kann:

- Namespace-Indizes aufgrund von Namespace-Uri ermitteln
- Vordefinierte Events feuern
- Den Server-Status liefern

Alle Angaben werden vom Anwender statisch in einem Datenobjekt hinterlegt.

Die zum Betrieb benötigten Daten werden in intern allokierten Speichern gehalten. Der Speicherbedarf ist optimiert, d.h. es wird nur der benötigte Speicher angefordert.

Über Funktionen erhält der Anwender lesenden Zugriff auf alle internen Daten wie z.B. Fehler-Codes.

4.3.2.1 Allgemeines

Der RunServer besteht aus mehreren Teilen.

Das Datenobjekt muss im AS ausgefüllt werden. Es enthält die Parameter für die verschiedenen Funktionalitäten (siehe unten).

Der FB BrbUaRunServerInit muss im Init-Teil des Tasks aufgerufen werden. Er liest das Datenobjekt aus, allokiert den benötigten Speicher und speichert darin die ausgelesenen Werte.

Der FB BrbUaRunServerCyclic muss im Cyclic-Teil des Tasks aufgerufen werden. Er ermittelt Namespaces und liest den Server-Status aus.

Der FB BrbUaRunServerExit muss im Exit-Teil des Tasks aufgerufen werden. Er gibt den allokierten Speicher wieder frei.

Eine zentrale Struktur-Variable enthält die gesamten Daten zum Betrieb des RunServers. Sie wird beim Aufruf jedes FB's als Zeiger übergeben.

4.3.2.2 Performance und Speicher-Verbrauch

Der RunServer ist sehr performant gestaltet und es sollte eigentlich nicht zu Zykluszeit-Verletzungen kommen. Mit der Anzahl der Funktionalitäten/Datenpunkte steigt allerdings auch die benötigte CPU-Last. So ist vom Anwender zu ermitteln, in welcher Task-Klasse/Zykluszeit der Client-Task ausreichend schnell lauffähig ist.

Die empfohlene Taskklasse ist #8. Es kann je nach Zielsystem und Anforderung aber auch eine andere Taskklasse gewählt werden.

Die empfohlene Zykluszeit ist 10ms. Es kann je nach Zielsystem und Anforderung aber auch eine andere Zykluszeit gewählt werden.

Achtung: Es ist darauf zu achten, ein Zielsystem mit genügend CPU-Leistung für die vom Anwender vorgegebenen Funktionalitäten zu verwenden!

Auch der Speicher-Verbrauch ist optimiert. So wird nur der Speicher allokiert, der auch benötigt wird. Bestimmte Funktionalitäten benötigen auch einiges an Arbeits-Speicher. Er wird mittels der Funktion TMP_alloc der Bibliothek SYS_Lib allokiert und liegt daher im System-RAM.

Achtung: Es ist darauf zu achten, ein Zielsystem mit genügend Arbeits-Speicher für die vom Anwender vorgegebenen Funktionalitäten zu verwenden!

4.3.2.3 Datenobjekt

Das Datenobjekt kann einen beliebigen Namen mit **max. 10 Zeichen** tragen. Es wird im AS vom Anwender befüllt und beinhaltet sämtliche Werte für die auszuführenden Funktionalitäten. Es wird im AS automatisch mitkompiliert und als Binär-Datei auf das Zielsystem übertragen (natürlich nur, wenn es der SW-Configuration unter ‚DataObjects‘ zugewiesen wurde).

Achtung: Wird nur das Datenobjekt geändert, aber nicht der RunServer-Task, muss ein Warmstart ausgeführt werden, da das Datenobjekt nur im Init ausgelesen wird!

4.3.2.3.1 Allgemeines

Um die fehlerfreie Funktionalität des RunServers zu gewährleisten, muss beim Ausfüllen des Datenobjekts folgendes beachtet werden:

- Die unten beschriebene Syntax muss unbedingt eingehalten werden
- Kommentare beginnen mit einem Semikolon ;
- Kommentare werden nicht kompiliert
- Ein einzelner Eintrag muss immer in einer Zeile stehen
- Ein einzelner Eintrag muss zwischen zwei Hoch-Komma „“ eingeschlossen sein
- Ein einzelner Eintrag beginnt immer mit einem Schlüsselwort, gefolgt von den Parametern
- Ein Schlüsselwort beginnt immer mit dem Paragraphen-Zeichen § und endet mit :
- Die Reihenfolge der einzelnen Einträge ist in vielen Fällen entscheidend (siehe unten)
- Leerzeilen zwischen den Einträgen werden ignoriert
- Die Parameter in einem Eintrag müssen durch ein Komma , getrennt sein
- Ein Parameter-Name endet immer mit einem =
- Die Reihenfolge der Parameter innerhalb eines Eintrags muss **unbedingt** eingehalten werden
- Leerzeichen/Tabs zwischen den Parameter werden ignoriert

Die einzelnen Einträge sind optional. Eingetragen sollte nur das sein, was die Anwendung auch benötigt. Nachfolgend werden nur die Syntax und die Bedeutung der Schlüsselwörter/Parameter erklärt. Der Zugriff auf die eingelesenen bzw. ermittelten Werte zur applikativen Verwendung folgt weiter unten.

Einige der Parameter eines Eintrags können auch optional sein. Sie werden dann entweder mit einem Default-Wert besetzt oder nicht beachtet.

Die Syntax von anzugebenden Variablen-Namen entspricht der Angabe

`<AppModul>::<Task>:<Variable>.<Element>`

Diese ist auch in der AS-Hilfe beschrieben (GUID = 28c9e872-0274-444d-8b4d-0aefb5bad3f6).

4.3.2.3.2 Namespaces

Es können für beliebig viele Namespace-Uri's die Namespace-Indizes ermittelt werden.

Diese Einträge sollten als allererstes im Datenobjekt stehen, da folgende Einträge u.U. von dieser Tabelle Gebrauch machen (siehe unten).

Die Syntax ist folgende:

`"$NAMESPACE: Uri=NamespaceUri"`

Beispiele:

`"$NAMESPACE: Uri=http://opcfoundation.org/UA/"`

`"$NAMESPACE: Uri=http://opcfoundation.org/UA/DI/"`

`"$NAMESPACE: Uri=http://br-automation.com/OpcUa/PLC/"`

`"$NAMESPACE: Uri=http://br-automation.com/OpcUa/PLC/PV/"`

Parameter	Beschreibung	Pflicht	Default-Wert
Uri=	Namespace-Uri, zu welchem ein Server-Namespace-Index ermittelt werden soll.	Ja	

Die Parameter entsprechen der Beschreibung in der AS-Hilfe zum FB

`AsOpcUas.UaSrv_GetNamespaceIndex.`

Namespace-Indizes, welche im Datenobjekt bei folgenden Einträgen verwendet werden, beziehen sich auf die Reihenfolge dieser 0-basierten Liste. Muss z.B. später ein Node aus dem B&R-PV-Namespace angegeben werden, so wäre es hier der Index 3, weil er als vierter Eintrag angegeben ist. Der Index wird dann zur Laufzeit mit dem am Server gültigen Index ersetzt. Wird ein Index verwendet, der im Datenmodul nicht angegeben ist, so wird er mit 0 ersetzt.

4.3.2.3.3 Events + EventFields

Hier ist es möglich, beliebig viele Events mit beliebig vielen Fields anzugeben, welche später ohne viel Aufwand gefeuert werden können.

Hier wird nur die Syntax eines Events bzw. Fields erklärt. Werden mehrere Events/Fields benötigt, müssen sie nur nacheinander im Datenobjekt eingetragen sein.

Die Syntax ist folgende:

```
"$EVENT: TypeNs=TypeDatObjNamespaceIndex, TypeId=TypeNodeIdentifizier"
"$SEVTFIELD: Name=FieldName, Var= VariablenName"
```

Beispiel:

```
"$EVENT: TypeNs=0, TypeId=2311"
"$SEVTFIELD: Name=Message, Var=:BrbUaSrvC:SrvTransitionEventData.Message"
"$SEVTFIELD: Name=Severity, Var=:BrbUaSrvC:SrvTransitionEventData.nSeverity"
"$SEVTFIELD: Name=/SourceName, Var=:BrbUaSrvC:SrvTransitionEventData.sSourceName"
"$SEVTFIELD: Name=/FromState, Var=:BrbUaSrvC:SrvTransitionEventData.FromState"
"$SEVTFIELD: Name=/FromState/Id, Var=:BrbUaSrvC:SrvTransitionEventData.nFromStateId"
"$SEVTFIELD: Name=/0:ToState, Var=:BrbUaSrvC:SrvTransitionEventData.ToState"
"$SEVTFIELD: Name=/0:ToState/0:Id, Var=:BrbUaSrvC:SrvTransitionEventData.nToStateId"
"$SEVTFIELD: Name=/Transition, Var=:BrbUaSrvC:SrvTransitionEventData.Transition"
"$SEVTFIELD: Name=/Transition/Id, Var=:BrbUaSrvC:SrvTransitionEventData.nTransitionId"
```

Event

Parameter	Beschreibung	Pflicht	Default-Wert
TypeNs=	Der Datenobjekt-Namespace-Index des Nodes, der den zu sendenden Event-Typen angibt (siehe Namespaces oben).	Nein	0
TypeId=	Der NodeIdentifizier des Nodes, der den zu sendenden Event-Typen angibt, z.B. 2311 für den Typ <code>TransitionEventType</code> . Es wird automatisch erkannt, ob es sich um eine Numeric- oder String-Adressierung handelt.	Nein	2041 (=BaseEventType)

Die Parameter entsprechen der Beschreibung in der AS-Hilfe zum FB `AsOpcUas.UaSrv_FireEvent`.

EventField

Parameter	Beschreibung	Pflicht	Default-Wert
Name=	Der Browse-Pfad zum Field. Einzelne Elemente werden durch / getrennt. Das / am Beginn muss immer vorhanden sein. Der Namespace-Index eines Pfad-Elements ist optional. Er muss dem Datenobjekt-Namespace-Index entsprechen (siehe Namespaces oben). Ist keiner angegeben, ist er 0.	Ja	
Var=	Der Name der Variablen, von der der gesendete Field-Wert genommen wird.	Ja	

Alle hier gelisteten Fields können dann von der Applikation besetzt und vom Client abonniert werden.

Die Parameter entsprechen der Beschreibung in der AS-Hilfe zum FB `AsOpcUac.UaSrv_FireEvent`.

4.3.2.4 Funktionsbausteine zum Betrieb des RunServers

Zum Betrieb des RunServers müssen verschiedene FB's aufgerufen werden.

4.3.2.4.1 BrbUaRunServerInit

```
void BrbUaRunServerInit(struct BrbUaRunServerInit* inst)
```

Argumente:

```
struct BrbUaRunServerInit* inst
    Zeiger auf die Funktionsblock-Instanz
```

Eingänge:

```
struct BrbUaRunServer_TYP* pRunServer
    Zeiger auf die RunServer-Struktur (genaue Beschreibung siehe unten)
```

Ausgänge:

```
UINT nStatus
    Funktionsblock-Status
    eBRB_OK = 0
    eBRB_ERR_NULL_POINTER
    eBRB_ERR_BUSY
    1..65534 = Fehlermeldung. Wenn es sich um einen System-Fehler (<50000) handelt, kann die Beschreibung dazu
    in der Studio-Hilfe gefunden werden.
```

Beschreibung:

Dieser FB **muss** im Init-Teil des Task aufgerufen werden.

Achtung: Wird er im zyklischen Teil aufgerufen, kann es zu Zykluszeit-Verletzung kommen!

Er liest das Datenobjekt aus, allokiert den benötigten Speicher und speichert darin die ausgelesenen Werte.

Wichtig: Da der Init nur einen Zyklus durchlaufen wird, der FB aber asynchron arbeitet, muss er in einer Schleife aufgerufen werden, bis der Status \neq eBRB_ERR_BUSY meldet.

Folgende Daten müssen vor dem Aufruf in der RunServer-Struktur (genaue Beschreibung siehe unten) gesetzt werden:

-RunServer.Cfg.sCfgDataObjName Der Name des Datenobjekts (max. 10 Zeichen)

4.3.2.4.2 BrbUaRunServerCyclic

```
void BrbUaRunServerCyclic(struct BrbUaRunServerCyclic* inst)
```

Argumente:

```
struct BrbUaRunServerCyclic* inst
    Zeiger auf die Funktionsblock-Instanz
```

Eingänge:

```
struct BrbUaRunServer_TYP* pRunServer
    Zeiger auf die RunServer-Struktur (genaue Beschreibung siehe unten)
```

Ausgänge:

```
UINT nStatus
    Funktionsblock-Status
    eBRB_OK
    eBRB_ERR_NULL_POINTER
    eBRB_ERR_INVALID_PARAMETER
    eBRB_ERR_BUSY
    1..65534 = Fehlermeldung. Wenn es sich um einen System-Fehler (<50000) handelt, kann die Beschreibung dazu
    in der Studio-Hilfe gefunden werden.
```

Beschreibung:

Dieser FB **muss** zyklisch aufgerufen werden.

Er ermittelt die Namespace-Indizes und aktualisiert den Server-Status, erledigt also alle Dinge, die erst im zyklischen Teil erledigt werden können.

Wurde vorher nicht der Init-FB aufgerufen, wird der Status eBRB_ERR_INVALID_PARAMETER zurückgegeben.

4.3.2.4.3 BrbUaRunServerExit

```
void BrbUaRunServerExit(struct BrbUaRunServerExit* inst)
```

Argumente:

```
struct BrbUaRunServerExit* inst
    Zeiger auf die Funktionsblock-Instanz
```

Eingänge:

```
struct BrbUaRunServer_TYP* pRunServer
    Zeiger auf die RunServer-Struktur (genaue Beschreibung siehe unten)
```

Ausgänge:

```
UINT nStatus
    Funktionsblock-Status
    eBRB_OK
    eBRB_ERR_NULL_POINTER
    eBRB_ERR_UA_ERROR
```

eBRB_ERR_BUSY

1..65534 = Fehlermeldung. Wenn es sich um einen System-Fehler (<50000) handelt, kann die Beschreibung dazu in der Studio-Hilfe gefunden werden.

Beschreibung:

Dieser FB **muss** im Exit-Teil des Task aufgerufen werden.

Achtung: Wird er im zyklischen Teil aufgerufen, kann es zu Zykluszeit-Verletzung kommen!

Er gibt den allokierten Speicher wieder frei. Dies ist besonders wichtig, wenn der Task nach Änderung übertragen wird. Dabei wird zuerst der Exit des alten Tasks und dann der Init des neuen Tasks abgearbeitet (siehe AS-Hilfe).

Würde dieser FB im Exit nicht aufgerufen, bliebe der alte allokierte Speicher bestehen und es würde neu-er allokiert, führte also zu einem Speicherleck.

Wichtig: Da der Exit nur einen Zyklus durchlaufen wird, der FB aber asynchron arbeitet, muss er in einer Schleife aufgerufen werden, bis der Status <> eBRB_ERR_BUSY meldet.

4.3.2.5 RunServer-Struktur

Diese Struktur muss vom Anwender angelegt und jedem der FB's als Zeiger übergeben werden.

Sie enthält die gesamten Daten zum Betrieb des RunServers. Manche der Daten müssen vom Anwender vor Aufruf eines FB's gesetzt werden, andere werden automatisch ermittelt.

Aus Übersichtlichkeits-Gründen ist die Struktur in einige Unterstrukturen aufgeteilt. Die meisten dieser Unterstrukturen sind interne Daten und dürfen nicht verändert werden!

BrbUaRunServer_TYP		RunServer-Struktur zum Betreiben des RunServers
Cfg	BrbUaRsCfg_TYP	Parameter
Namespaces	BrbUaRsNamespaces_TYP	Daten der Namespaces
Events	BrbUaRsEvents_TYP	Daten der Events
State	BrbUaRsState_TYP	Status des RunServers

4.3.2.5.1 Cfg

BrbUaRsCfg_TYP		Parameter
sCfgDataObjName	STRING[nBRBUA_DATAOBJECT_NAME_CHAR_MAX]	Name des Datenobjekts

Der Parameter sCfgDataObjName gibt den Namen des Datenobjekts an (max. 10 Zeichen), in welchem die funktionalen Parameter hinterlegt sind. Er muss vom Anwender schon vor dem Aufruf des Init-FB's korrekt belegt sein.

4.3.2.5.2 Namespaces/Events

BrbUaRsNamespaces_TYP		Daten der Namespaces
nNamespaceCount	UINT	Anzahl der Namespaces
pNamespaces	UDINT	Interner Zeiger
nMemLen	UDINT	Interne Speichergröße
BrbUaRsEvents_TYP		Daten der Events
nEventCount	UINT	Anzahl der Subscriptions
pEvents	UDINT	Interner Zeiger
nMemLen	UDINT	Interne Speichergröße
nFieldsCountTotal	UDINT	Anzahl aller Fields aller Events

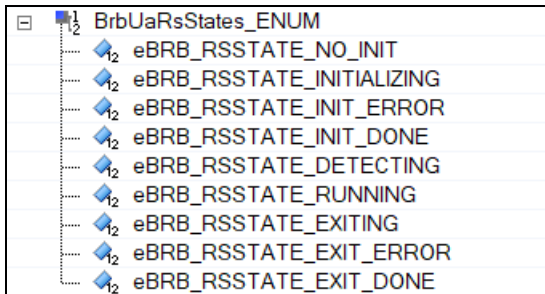
Diese Unterstrukturen enthalten jeweils die Anzahl der vom Datenobjekt eingelesenen Elemente.

Die restlichen Daten darin dienen nur internen Zwecken. Diese Daten dürfen vom Anwender auf keinen Fall verändert werden! Der Anwender-Zugriff auf die intern gehaltenen Daten geschieht über Funktionen (siehe weiter unten).

4.3.2.5.3 State

BrbUaRsState_TYP		Status des Clients und der Verbindung und des Servers
eState	BrbUaRsStates_ENUM	Status des RunServers
nErrorId	UDINT	OpCua-Fehler-Status bei internen Aufrufen
sErrorText	STRING[nBRBUA_VALUE_TEXT_CHAR_MAX]	Details zum Fehler
eServerState	UAServerState	Status des Servers

4.3.2.5.3.1 eState



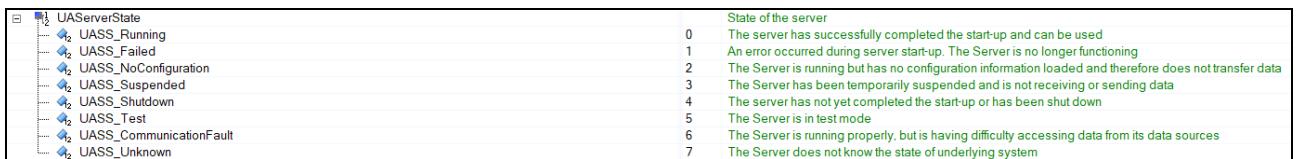
1	BrbUaRsStates_ENUM
2	eBRB_RSSTATE_NO_INIT
3	eBRB_RSSTATE_INITIALIZING
4	eBRB_RSSTATE_INIT_ERROR
5	eBRB_RSSTATE_INIT_DONE
6	eBRB_RSSTATE_DETECTING
7	eBRB_RSSTATE_RUNNING
8	eBRB_RSSTATE_EXITING
9	eBRB_RSSTATE_EXIT_ERROR
10	eBRB_RSSTATE_EXIT_DONE

Dieser Enumerations-Wert stellt die verschiedenen Phasen vom Init über Cyclic bis zum Exit dar. Alle Schritte werden durch Aufruf der FB's vom Anwender angestoßen.

4.3.2.5.3.2 nErrorId/sErrorText

Tritt beim internen Aufruf eines FB'S der System-Bibliothek `AsOpcUas` ein Fehler auf, so wird dieser als `OpcUa-Fehler-Status` auf das Element `nErrorId` übernommen und `sErrorText` enthält einen Klartext, bei welcher Gelegenheit der Fehler auftrat (z.B. beim Ermitteln eines Namespace-Indizes). Wenn möglich werden auch die 0-basierten Indizes der betroffenen Elemente angegeben. Die Indizes beziehen sich dabei immer auf die Reihenfolge im Datenobjekt.

4.3.2.5.3.3 eServerState



UAServerState	State of the server
UASS_Running	0 The server has successfully completed the start-up and can be used
UASS_Failed	1 An error occurred during server start-up. The Server is no longer functioning
UASS_NoConfiguration	2 The Server is running but has no configuration information loaded and therefore does not transfer data
UASS_Suspended	3 The Server has been temporarily suspended and is not receiving or sending data
UASS_Shutdown	4 The server has not yet completed the start-up or has been shut down
UASS_Test	5 The Server is in test mode
UASS_CommunicationFault	6 The Server is running properly, but is having difficulty accessing data from its data sources
UASS_Unknown	7 The Server does not know the state of underlying system

Diese Enumeration enthält den vom intern aufgerufenen FB `UaSrv_GetServerState` ermittelten Server-Status (siehe AS-Hilfe). Bei einer B&R-Sps ist er normalerweise immer `Running`.

4.3.2.6 Ausführen von vordefinierten Funktionen

Mit diesen FB's ist es möglich, im Datenobjekt vordefinierte Funktionen auszuführen. Die dabei anzugebenden Indizes beziehen sich auf die Reihenfolge im Datenobjekt. Das erste Element hat den Index 0.

4.3.2.6.1 BrbUaRsFireEvent

```
void BrbUaRsFireEvent(struct BrbUaRsFireEvent* inst)
```

Argumente:

```
struct BrbUaRsFireEvent* inst  
Zeiger auf die Funktionsblock-Instanz
```

Eingänge:

```
struct BrbUaRunClient_TYP* pRunServer  
Zeiger auf die RunServer-Struktur (genaue Beschreibung siehe oben)  
UINT nEventIndex  
Der 0-basierte Index des Events laut Reihenfolge im Datenobjekt
```

Ausgänge:

```
BOOL bInit  
Vor dem Feuern für einen Zyklus auf 1 zum optionalen Besetzen der Fields  
UINT nStatus  
Funktionsblock-Status  
eBRB_OK  
eBRB_ERR_NULL_POINTER  
eBRB_ERR_UA_ERROR
```



```
eBRB_ERR_UA_NO_ELEMENTS
eBRB_ERR_UA_INVALID_INDEX
eBRB_ERR_UA_NOT_RUNNING
eBRB_ERR_BUSY
1..65534 = Fehlermeldung. Wenn es sich um einen System-Fehler (<50000) handelt, kann die Beschreibung dazu
in der Studio-Hilfe gefunden werden.
UINT nErrorId
    OpcUa-Fehler-Status
STRING sErrorId
    OpcUa-Fehler-Status als Text
UINT nFireCount
    Anzahl der gefeuerten Events
```

Beschreibung:

Feuert das angegebene Event durch den internen Aufruf von `AsOpcUas.UaSrv_FireEvent`. Die Field-Werte werden von den gemappten Variablen genommen.

4.3.2.7 Zugriff auf die internen Daten

Auf alle intern im allokierten Speicher gehaltenen Daten (z.B. vom Datenobjekt eingelesene Parameter oder ermittelte Daten) kann der Anwender über eigene Funktionen lesend zugreifen. Diese Daten (z.B. NamespaceIndex) können dann applikativ verwendet werden.

Die lesenden Funktionen (`BrbUaRsGetXXX`) arbeiten mit reinem Speicherzugriff und sind daher sehr performant.

Ist beim Ermitteln der benötigten Daten (z.B. beim Ermitteln eines Namespaces) ein Fehler aufgetreten, so kann dieser damit eingesehen werden.

Achtung: Die internen Daten werden bei diesen Funktionen immer nur auf den übergebenen Zeiger kopiert. Eine Änderung in dieser Kopie bewirkt also keine Änderung an den internen Daten, also auch keine Parameter-Änderung!

4.3.2.7.1 BrbUaRsGetNamespace

```
unsigned short BrbUaRsGetNamespace(struct BrbUaRunServer_TYP* pRunServer, unsigned short
nDatObjNamespaceIndex, struct BrbUaRsNamespace_TYP* pServerNamespace)
```

Argumente:

```
struct BrbUaRunServer_TYP* pRunServer
    Zeiger auf die RunServer-Struktur (genaue Beschreibung siehe oben)
UINT nDatObjNamespaceIndex
    Der 0-basierte Index des Namespaces laut Reihenfolge im Datenobjekt
struct BrbUaRsNamespace_TYP* pServerNamespace
    Zeiger auf eine Struktur, die die Werte aufnimmt. Kann auch 0 sein
```

Rückgabe:

```
UINT
    Der serverseitige Namespace-Index
```

Beschreibung:

Gibt den Server-Namespace-Index aufgrund des Namespace-Index aus dem Datenobjekt zurück. Ist der Namespace nicht vorhanden, so wird 0 zurückgegeben.

Optional kann zusätzlich ein Zeiger auf folgende Struktur übergeben werden:

BrbUaRsNamespace_TYP			Daten eines Namespaces
sNamespaceUri	STRING[nBRBUA_NAMESPACE_URI_CHAR_MAX]		Namespace-Uri aus dem Datenobjekt
nNamespaceIndex	UINT		Server-Namespace-Index
nErrorId	DWORD		OpcUa-Fehler-Status

Ist der Zeiger angegeben, wird diese Struktur befüllt. Hier kann auch erkannt werden, ob und welcher Fehler beim Ermitteln des Server-Namespace-Indizes aufgetreten ist (z.B. bei Angabe einer nicht vorhandenen Uri).

4.3.2.7.2 BrbUaRsGetEvent

```
unsigned short BrbUaRsGetEvent(struct BrbUaRunServer_TYP* pRunServer, unsigned short nEventIndex,
struct BrbUaRsEvent_TYP* pEvent, struct BrbUaRsEventIntern_TYP* pEventIntern)
```

Argumente:

```
struct BrbUaRunServer_TYP* pRunServer
    Zeiger auf die RunServer-Struktur (genaue Beschreibung siehe oben)
UINT nEventIndex
    Der 0-basierte Index des Events laut Reihenfolge im Datenobjekt
struct BrbUaRsEvent_TYP* pEvent
    Zeiger auf eine Struktur, die die Werte aufnimmt.
struct BrbUaRsEventIntern_TYP* BrbUaRsEventIntern
    Optionaler Zeiger auf eine Struktur, die auch die internen Werte aufnimmt. Kann auch 0 sein
```

Rückgabe:

```
UINT
    eBRB_OK
    eBRB_ERR_NULL_POINTER
    eBRB_ERR_UA_NO_ELEMENTS
    eBRB_ERR_UA_INVALID_INDEX
```

Beschreibung:

Gibt die ermittelten Daten zu einem im Datenobjekt angegebenen Event zurück.
Dazu muss ein Zeiger auf folgende Struktur übergeben werden:

BrbUaRsEvent_TYP			Daten eines Events
nDatObjNamespaceIndex	UINT		Namespace-Index laut Datenobjekt
TypeNodeId	UANodeID		TypeNodeId laut Datenobjekt
nFieldCount	UINT		Anzahl der Fields

Hinweis: Nähere Beschreibungen zu den einzelnen Werten siehe AS-Hilfe zum FB UaSrv_FireEvent.

4.3.2.7.3 BrbUaRsGetEventField

```
unsigned short BrbUaRsGetEventField(struct BrbUaRunServer_TYP* pRunServer, unsigned short nEventIndex, unsigned short nEventFieldIndex, struct UaSrv_FireEventFieldType* pEventField)
```

Argumente:

```
struct BrbUaRunServer_TYP* pRunServer
    Zeiger auf die RunServer-Struktur (genaue Beschreibung siehe oben)
UINT nEventIndex
    Der 0-basierte Index des Events laut Reihenfolge im Datenobjekt
UINT nEventFieldIndex
    Der 0-basierte Index des EventFields laut Reihenfolge im Datenobjekt
struct UaSrv_FireEventFieldType * pEventField
    Zeiger auf eine Struktur, die die Werte aufnimmt.
```

Rückgabe:

```
UINT
    eBRB_OK
    eBRB_ERR_NULL_POINTER
    eBRB_ERR_UA_NO_ELEMENTS
    eBRB_ERR_UA_INVALID_INDEX
```

Beschreibung:

Gibt die ermittelten Daten zu einem im Datenobjekt angegebenen EventField zurück.
Dazu muss ein Zeiger auf folgende Struktur aus der System-Bibliothek AsOpcUac übergeben werden:

Bezeichnung	Datentyp	Beschreibung
Name	STRING [MAX_LENGTH_EVENTFIELDPATH]	BrowsePath des Feldes wie es im OPC-UA Informationsmodell definiert ist. Der BrowsePath setzt sich aus den Browse-Namen der Knoten zusammen. Beispiele: <ul style="list-style-type: none">• /EventId adressiert das Feld EventId• /Transition/Id bezieht sich auf das Feld Id des Event-Typs TransitionEventType
Variable	STRING[MAX_LENGTH_VARIABLE]	Name der Variable deren Wert dem Feld zugewiesen wird.
ErrorID	DWORD	Status der Zuweisung. Häufige Ursachen für einen Fehler sind: <ul style="list-style-type: none">• PlcOpen_BadEventFieldSelection: Angegebenes Feld existiert nicht• PlcOpen_BadVariableNameInvalid: Angegebene Variable existiert nicht• Bad_TypeMismatch: Datentyp des Feldes und der Variable nicht kompatibel• PlcOpen_BadInvalidEventFieldValue: Der Wert ist für das Feld nicht erlaubt.

Hinweis: Nähere Beschreibungen zu den einzelnen Werten siehe AS-Hilfe zum FB UaSrv_FireEvent.

4.3.2.8 BrbUaRsMonitor

```
unsigned short BrbUaRsMonitor(struct BrbUaRunServer_TYP* pRunServer, struct BrbUaRsMonitor_TYP* pMonitor)
```

Argumente:

```
struct BrbUaRunServer_TYP* pRunServer  
    Zeiger auf die RunServer-Struktur (genaue Beschreibung siehe oben)  
struct BrbUaRsMonitor_TYP* pMonitor  
    Zeiger auf die Monitor-Struktur
```

Rückgabe:

```
UINT  
    eBRB_OK  
    eBRB_ERR_NULL_POINTER  
    eBRB_ERR_NOT_ENABLED
```

Beschreibung:

Mit dieser Funktion kann sehr einfach ein Monitor zum Anzeigen der internen Daten des RunServers implementiert werden. Da hier auch evtl. aufgetretene Fehler-Codes ausgelesen werden können, wird dadurch die Entwicklung und Inbetriebnahme sehr erleichtert.

Dazu muss ein Zeiger auf folgende Struktur übergeben werden:

BrbUaRsMonitor_TYP		Monitor für interne Daten
bEnable	BOOL	1=Aktiv
nMonitorStatus	UINT	Status des Monitors
Namespace	BrbUaRsMonitorNamespace_TYP	Interne Daten zu einem Namespace
Event	BrbUaRsMonitorEvent_TYP	Interne Daten zu einem Event
EventField	BrbUaRsMonitorEventField_TYP	Interne Daten zu einem Field

Durch `bEnable` wird der Monitor aktiviert.

Gibt es ein grundsätzliches Problem, wird dies in `nMonitorStatus` (siehe auch Rückgabewert der Funktion) angegeben.

Jede dieser Unterstrukturen bietet mithilfe der oben beschriebenen Zugriffs-Funktionen die Anzeige der internen Daten eines bestimmten Elements, welches im Datenobjekt angegeben wurde (z.B. eines Events).

Bei jeder Unterstruktur kann der Index des Elements (bzw. mehrere Indizes bei Schachtelungen) angegeben werden. Die internen Werte des adressierten Elements werden dann auf die ebenfalls enthaltene Struktur kopiert. Ist dies nicht möglich (z.B. weil ein ungültiger Index angegeben wurde), wird der in der Unterstruktur befindliche `nMonitorStatus` gesetzt.

Die Beschreibung zu den Werten eines Elements siehe oben bei der entsprechenden Get-Funktion.

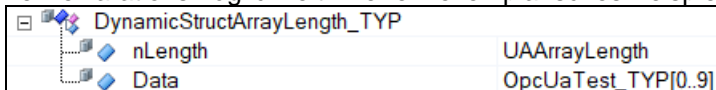
5 Anhänge

5.1 Hinweise zu dynamischen Arrays

Grundsätzlich ist jedes Array bei OpcUa dynamisch. Das bedeutet, dass immer nur so viel Elemente übertragen werden, wie aktuell gültig sind.

Arrays müssen auf einer Sps aber statisch im Editor des Automation Studio deklariert werden, also mit einer festen Anzahl von Elementen.

Um trotzdem mit (manchmal sehr hilfreichen) dynamischen OpcUa-Arrays arbeiten zu können, gibt es eine Deklarationsmöglichkeit. Hier ein exemplarisches Beispiel dazu:



Das Element ‚Data‘ enthält das Array mit einer statischen Deklaration (hier 10 Array-Elemente).

Das Element ‚nLength‘ gibt an, wie viele Elemente tatsächlich gültig sind. Natürlich darf die angegebene Länge die Anzahl von 10 nicht überschreiten!

Als Typ für ‚Data‘ dürfen Standard-Datentypen genauso verwendet werden wie Struktur-Typen (auch selbstdefinierte).

Unterstützt wird dieses Konzept bei Read/Write, Subscription und Methoden-Argumente von Client und Server.

Dabei wird eine Instanz-Variable des obigen Typs angelegt und bei der jeweiligen Funktion angegeben. Der OpcUa-Treiber erkennt sie als dynamisches Array und behandelt sie entsprechend.

Achtung: Was passiert, wenn die aktuelle Länge größer ist als das statische Array?

Wird z.B. ein Datenpunkt mit obiger Deklaration gelesen, der 11 Elemente beinhaltet, kann der Wert des Arrays nicht auf die Variable kopiert werden, da diese zu klein ist. Die Variable wird also nicht aktualisiert. Stattdessen wird der Fehler ‚8074000 = BadTypeMismatch‘ zurückgegeben.

Es sollte also immer mindestens die maximal erreichbare Element-Anzahl deklariert werden.

Auch die in einer System-Bibliothek deklarierten dynamischen Arrays können benutzerdefiniert abgeleitet und so mit einer passenden Max-Anzahl definiert werden.

5.2 Hinweise zu 64-Bit-Datentypen

Die größten nativ auf der Sps unterstützten Integer-Datentypen in IEC sind UDINT (Uint32) und DINT (Int32).







Manchmal werden aber auch größere Datentypen als Zähler benötigt. OpcUa stellt deshalb 64-Bit-Datentypen im Foundation-Modell bereit:

OpcUa-Datentyp	Adresse	Pfad	Werte-Bereich
Uint64	ns=0;i=9	/Root/Types/DataTypes/BaseDataType/Number/UInteger/UInt64	0...18446744073709551615
Int64	ns=0;i=8	/Root/Types/DataTypes/BaseDataType/Number/Integer/Int64	-9223372036854775808... +9223372036854775807

Die Unterstützung für 64-Bit-Werte (Uint64 und Int64) ist in der Basis-Bibliothek „BrbLib“ implementiert (Details siehe dortige Hilfe). Sps-seitig werden die Werte als Struktur-Instanz gehalten:



Um diese am OpcUa-Server tatsächlich als obige OpcUa-Datentypen zu übertragen, müssen sie in der *.uad-Datei in der Spalte „UA Datatype“ entsprechend parametrisiert werden:

Name		Datatype	UA Datatype
		BrbInt64_TYP	ns=0;i=8
		UDINT	
		UDINT	
		BrbUInt64_TYP	ns=0;i=9
		UDINT	
		UDINT	

Der Server behandelt die Strukturen dann wie den angegebenen OpcUa-Datentypen.

Dies funktioniert aber nur, wenn die Sps-Variable die Länge in Bytes hat, welche vom angegebenen OpcUa-Typ benötigt wird. Die OpcUa-Datentypen UInt64 bzw. Int64 benötigen jeweils 8 Byte, welche durch die Instanzen der Strukturen 'BrbUInt64_TYP' bzw. 'BrbInt64_TYP' bereitgestellt werden.

Achtung: In der Basis-Bibliothek „BrbLib“ sind Hinweise zur Konsistenz der 64-Bit-Datentypen auf der Sps enthalten, die unbedingt berücksichtigt werden sollten!