



BrbLib V6.00

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	6
1.1 Hinweise zum AS und Compiler	6
1.2 Abhängigkeiten	6
1.3 Hinweise zu StructuredText und anderen IEC-Sprachen.....	6
1.4 Getestet mit UnitTests	6
1.5 Geprüft mit ClangTidy	7
1.6 Quellcode und Binär-Variante der Bibliothek.....	7
1.6.1 Quellcode-Variante	7
1.6.2 Binär-Variante	8
1.7 Neueste Versionen auf GitHub	8
2 Revisionsgeschichte	9
2.1 BrbLib V6.00 – 2024-09-13.....	9
2.1.1 Portierung auf neuere Versionen	9
2.1.2 Hinweise auf AS.....	9
2.1.3 Erweiterung Schrittketten-Protokollierung um 2. Variante.....	9
2.1.4 Neue Funktion BrbGetTimeOfDay	9
3 Fehlernummern	10
3.1 eBRB_ERR_OK = 0.....	10
3.2 eBRB_ERR_NULL_POINTER = 50000	10
3.3 eBRB_ERR_INVALID_PARAMETER = 50001	10
3.4 eBRB_ERR_NOT_ENABLED = 50002	10
3.5 eBRB_ERR_OVERFLOW = 50003	10
3.6 eBRB_ERR_DIVISION_BY_0 = 50004	10
3.7 eBRB_ERR_STRING_TOO_SHORT = 50005.....	10
3.8 eBRB_ERR_STRING_TOO_LONG = 50006	10
3.9 eBRB_ERR_BUSY = 65535	10
4 Pakete	11
4.1 StepHandling.....	11
4.1.1 Variante 1.....	11
4.1.1.1 BrbStepHandler.....	12
4.1.1.2 BrbStartStepTimeout.....	13
4.1.1.3 BrbStopStepTimeout.....	14
4.1.2 Variante 2.....	14
4.1.2.1 BrbStepHandler2.....	15
4.1.2.2 BrbStartStepTimeout2.....	17
4.1.2.3 BrbStopStepTimeout2.....	17
4.1.3 Stoppuhr	17
4.1.3.1 BrbStartStopWatch.....	17
4.1.3.2 BrbStopStopWatch.....	18
4.2 TaskCommunication	19
4.2.1 BrbSetCaller.....	19
4.2.2 BrbClearDirectBox	20
4.2.3 BrbClearCallerBox	20
4.3 VarHandling	21
4.3.1 BrbSaveVarAscii	21
4.3.2 BrbLoadVarAscii	22
4.3.3 BrbSaveVarBin	23

4.3.4 BrbLoadVarBin	23
4.4 FileHandling	25
4.4.1 BrbCheckUsbSticks	25
4.4.2 BrbReadDir	25
4.4.3 BrbDeleteFiles	27
4.4.4 BrbLoadFileDataObj	28
4.4.5 BrbSaveFileDataObj	29
4.4.6 BrbLoadFileBin	29
4.4.7 BrbCheckFileName	30
4.4.8 BrbCheckFileEnding	30
4.4.9 BrbCombinePath	30
4.5 Logger	32
4.5.1 BrbLoggerReadHierarchicalList	32
4.6 TimeAndDate	36
4.6.1 BrbSetDtStruct	36
4.6.2 BrbSetDt	36
4.6.3 BrbSetTimespan	37
4.6.4 BrbSetTimespanT	37
4.6.5 BrbGetTimeText	38
4.6.6 BrbGetCurrentTimeText	39
4.6.7 BrbGetTimeTextDtStruct	39
4.6.8 BrbGetTimeTextDt	40
4.6.9 BrbGetDtFromTimeText	40
4.6.10 BrbRtcTimeToDtStruct	40
4.6.11 BrbDtStructCompare	41
4.6.12 BrbDtStructAddDays	41
4.6.13 BrbDtStructAddHours	41
4.6.14 BrbDtStructAddMinutes	42
4.6.15 BrbDtStructAddSeconds	42
4.6.16 BrbDtStructAddMilliseconds	42
4.6.17 BrbDtStructAddMillisecondsLReal	43
4.6.18 BrbGetTimeTextMs	43
4.6.19 BrbGetTimeDiffMsDtStruct	44
4.6.20 BrbGetWeekdayDtStruct	44
4.6.21 BrbGetTimeOfDay	44
4.6.22 BrbGetWeekdayDt	45
4.6.23 BrbTimerSwitch	45
4.7 Strings + WcStrings	48
4.7.1 BrbWcCopyStringtoWString	48
4.7.2 BrbWcCopyWStringtoString	48
4.7.3 BrbUsintToHex	48
4.7.4 BrbUsintArrayToHex	49
4.7.5 BrbHexToUsintArray	49
4.7.6 BrbUdintToAscii	49
4.7.7 BrbAsciiToUdint	50
4.7.8 BrbUdintToHex	50
4.7.9 BrbHexToUdint	50
4.7.10 BrbUdintToBin	50
4.7.11 BrbDintToHex	51
4.7.12 BrbHexToDint	51
4.7.13 BrbAsciiFieldToString	52
4.7.14 BrbStringGetIndexOf + BrbWcStringGetIndexOf	53
4.7.15 BrbStringGetLastIndexOf + BrbWcStringGetLastIndexOf	53
4.7.16 BrbStringGetAdrOf + BrbWcStringGetAdrOf	53
4.7.17 BrbStringGetLastAdrOf + BrbWcStringGetLastAdrOf	54
4.7.18 BrbStringStartsWith + BrbWcStringStartsWith	54
4.7.19 BrbStringEndsWith + BrbWcStringEndsWith	54
4.7.20 BrbStringGetSubText + BrbWcStringGetSubText	54
4.7.21 BrbStringGetSubTextByLen + BrbWcStringGetSubTextByLen	55
4.7.22 BrbStringGetSubTextByAdr + BrbWcStringGetSubTextByAdr	55
4.7.23 BrbStringAppend + BrbWcStringAppend	55
4.7.24 BrbStringCut + BrbWcStringCut	56

4.7.25 BrbStringCutFromLastSeparator.....	56
4.7.26 BrbStringInsert + BrbWcStringInsert.....	56
4.7.27 BrbStringReplace + BrbWcStringReplace.....	57
4.7.28 BrbStringPadLeft + BrbWcStringPadLeft.....	57
4.7.29 BrbStringPadRight + BrbWcStringPadRight.....	57
4.7.30 BrbStringTrimLeft + BrbWcStringTrimLeft.....	57
4.7.31 BrbStringTrimRight + BrbWcStringTrimRight.....	58
4.7.32 BrbStringSplit.....	58
4.7.33 BrbStringSplitEmpty.....	58
4.7.34 BrbStringConvertRealFromExp.....	59
4.7.35 BrbStringConvertRealToExp.....	59
4.7.36 BrbStringFormatFractionDigits.....	59
4.7.37 BrbStringSwap.....	60
4.7.38 BrbStringToUpper + BrbWcStringToUpper.....	60
4.7.39 BrbStringToLower + BrbWcStringToLower.....	60
4.7.40 BrbStringIsNumerical + BrbWcStringIsNumerical.....	60
4.7.41 BrbStringIsHex + BrbWcStringIsHex.....	61
4.7.42 BrbStringCountText + BrbWcStringCountText.....	61
4.7.43 BrbStringRepeat + BrbWcStringRepeat.....	61
4.7.44 BrbStringCopy + BrbWcStringCopy.....	62
4.7.45 BrbStringCat + BrbWcStringCat.....	63
4.8 BigDatatypes.....	64
4.8.1 UInt64 (ULINT).....	64
4.8.1.1 Struktur.....	64
4.8.1.2 BrbUInt64FromString.....	64
4.8.1.3 BrbUInt64ToString.....	64
4.8.1.4 BrbUInt64Compare.....	65
4.8.1.5 BrbUInt64Inc.....	65
4.8.1.6 BrbUInt64Dec.....	65
4.8.1.7 BrbUInt64Calculate.....	65
4.8.2 Int64 (LINT).....	66
4.8.2.1 Struktur.....	66
4.8.2.2 BrbInt64FromString.....	66
4.8.2.3 BrbInt64ToString.....	66
4.8.2.4 BrbInt64Compare.....	67
4.8.2.5 BrbInt64Inc.....	67
4.8.2.6 BrbInt64Dec.....	67
4.8.2.7 BrbInt64Calculate.....	67
4.8.3 Hilfsfunktionen.....	68
4.8.3.1 BrbIntegerStringAdd.....	68
4.8.3.2 BrbByteArrayAdd.....	68
4.8.3.3 BrbByteArraySubtract.....	69
4.8.4 Hinweise zur Konsistenz.....	69
4.9 Xml.....	70
4.9.1 BrbXmlGetTagText.....	70
4.9.2 BrbXmlGetNextTag.....	71
4.10 Memory.....	72
4.10.1 MemList.....	72
4.10.1.1 Struktur.....	72
4.10.1.2 BrbMemListClear.....	72
4.10.1.3 BrbMemListIn.....	72
4.10.1.4 BrbMemListOut.....	72
4.10.1.5 BrbMemListGetEntry.....	73
4.10.2 Fifo.....	73
4.10.2.1 BrbFifoIn.....	73
4.10.2.2 BrbFifoOut.....	73
4.10.3 Lifo.....	74
4.10.3.1 BrbLifoIn.....	74
4.10.3.2 BrbLifoOut.....	74
4.10.4 Bit-Funktionen.....	74
4.10.4.1 BrbGetBitUdint.....	74
4.10.4.2 BrbSetBitUdint.....	75

4.10.4.3 BrbGetBitMaskUdint	75
4.10.4.4 BrbSetBitMaskUdint	75
4.10.4.5 BrbGetBitUint	75
4.10.4.6 BrbSetBitUint	76
4.10.4.7 BrbGetBitMaskUint	76
4.10.4.8 BrbSetBitMaskUint	76
4.10.4.9 BrbGetBitUsint	76
4.10.4.10 BrbSetBitUsint	77
4.10.4.11 BrbGetBitMaskUsint	77
4.10.4.12 BrbSetBitMaskUsint	77
4.10.5 ByteArray-Funktionen	78
4.10.5.1 BrbGetByteArrayBit	78
4.10.5.2 BrbSetByteArrayBit	78
4.10.5.3 BrbSetByteArrayBits	79
4.11 Math	80
4.11.1 BrbAbsReal	80
4.11.2 BrbAbsLReal	80
4.11.3 BrbIsNearlyZeroReal	80
4.11.4 BrbIsNearlyZeroLReal	80
4.11.5 BrbCompareReal	80
4.11.6 BrbCompareLReal	81
4.11.7 BrbIsWithinRangeDint	81
4.11.8 BrbIsWithinRangeUdint	81
4.11.9 BrbIsWithinRangeReal	82
4.11.10 BrbIsWithinRangeLReal	82
4.11.11 BrbGetAngleRad	82
4.11.12 BrbGetAngleDeg	82
4.11.13 BrbNormalizeAngleRad	82
4.11.14 BrbNormalizeAngleDeg	83
4.11.15 BrbIsWithinRangeAngle	83
4.11.16 BrbDetectAngleTransition	83
4.11.17 BrbGetDistance2d	84
4.11.18 BrbRoundDint	84
4.11.19 BrbScaleLReal	85
4.11.20 BrbScaleAnalogInput	86
4.11.21 BrbScaleAnalogOutput	86
4.12 Random	88
4.12.1 BrbGetRandomPercent	88
4.12.2 BrbGetRandomBool	88
4.12.3 BrbGetRandomUdint	88
4.12.4 BrbGetRandomDint	88
4.12.5 BrbGetRandomText	89
4.12.6 BrbGetRandomString	89
4.12.7 BrbGetRandomStringExt	90
4.13 IpAndSubnet	91
4.13.1 BrbIpToNumericalArray	91
4.13.2 BrbCheckIpAddress	91
4.13.3 BrbCheckSubnet	91
4.13.4 BrbGetIpAddressType	92
4.13.5 BrbCheckIpEqualSubnet	93
4.14 Additional	94
4.14.1 BrbDebounceInput	94
4.14.2 BrbGetStructureMemberOffset	94
4.14.3 BrbGetCompilerVersion	95

1 Allgemeines

Die Bibliothek „BrbLib“ enthält viele nützliche Funktionen in Bereichen wie Strings, Schrittketten, Speicherverwaltung etc. 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 AS und Compiler

Das Entwicklungs- und Demo-Projekt wird mit AS6.00 und Compiler V11.3.0 entwickelt und getestet. Die Bibliothek kann aber auch von AS4.9 bis AS4.12 unter Compiler V6.3.0 eingesetzt werden.

1.2 Abhängigkeiten

Es besteht eine Abhängigkeit von folgenden Bibliotheken:

- AsBrStr
- AsBrWstr
- ArEventLog
- AsTime
- AsUSB
- DataObj
- FileIO
- Standard
- Sys_lib

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.

Einschränkung:

Bei manchen Funktionsblöcken sind optional über sogenannte Funktionszeiger benutzerdefinierte Erweiterungen implementiert. Beispiel: Beim FB ‚BrbReadDir‘ kann die Standard-Filterung benutzerdefiniert erweitert werden.

Da die IEC-Sprachen keine Funktionszeiger unterstützen, sind diese Erweiterungen nur in ANSI-C nutzbar. Die entsprechenden Eingänge des FB’s für die Funktionszeiger müssen in IEC-Sprachen auf 0 gesetzt werden. Ansonsten können auch diese FB’s ohne Probleme 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 (Casting)	Zuweisung eines UDINT an eine UINT-Variable kann zu Datenverlust führen	Einfügen expliziter, also gewollter 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 Quellcode und Binär-Variante der Bibliothek

Die Binär-Variante der Bibliothek ist ab V5.04 nicht mehr im Release enthalten, weil es zu viele Kombinationen (Zielsystem SG4 oder SGC, Prozessor Intel oder ARM, eingestellte Compiler-Version usw.) gibt, die Einfluss auf das Kompilat haben.

Außerdem ist es für den Anwender sehr leicht möglich, die benötigte Binär-Variante selbst zu erstellen (siehe AS-Hilfe GUID d750bdd3-0aad-4486-8c0d-4eb43372b325).

Welche Variante der Anwender in seinem Projekt verwendet, sollte von diesen Punkten abhängig gemacht werden:

1.6.1 Quellcode-Variante

Sie enthält den kompletten Quellcode aller Funktionen in ANSI-C. Somit kann der Anwender diesen studieren und unter Umständen eine ähnliche/abgewandelte Funktion sehr leicht in einer eigenen Bibliothek implementieren. Auch das Online-Debuggen durch Breakpoints ist möglich.

Beim Rebuild wird allerdings auch diese Bibliothek nochmals kompiliert. Dies kann je nach verwendetem Rechner einige Zeit in Anspruch nehmen.

Hinweis: Von der Änderung der Funktionen in der ausgelieferten Bibliothek wird abgeraten, da dann ein Umstieg auf eine neuere Version schwierig bis unmöglich wird.

1.6.2 Binär-Variante

Sie enthält nur vorkompilierte Module der Bibliothek für einen bestimmten Prozessor und Compiler. Es ist also kein Quellcode enthalten. Der Vorteil besteht darin, dass die Bibliothek auch bei einem Rebuild nicht mehr kompiliert werden muss. Dies bedeutet unter Umständen einen großen Zeitvorteil.

Achtung: Bei einer für ARM-Prozessoren exportierten Binär-Bibliothek kann nicht in den ArSim-Modus geschaltet werden, da ArSim wiederum die Intel-Version benötigt. Soll ArSim verfügbar sein, muss die Quellcode-Variante der Bibliothek eingefügt werden, denn nur dann kann sie je nach Prozessor kompiliert werden.

1.7 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).

Die 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, OpcUa-Publisher

Es ist unter diesem Link erhältlich:

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

Außerdem gibt es ein Beispiel-Projekt für OpcUa inklusive der Bibliothek BrbLibUa:

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

2 Revisionsgeschichte

Ab V5.01 ist hier nur die letzte Version erwähnt. Die gesamte Revisionsgeschichte wurde in die Datei „BrbLib Revisionsgeschichte“ ausgelagert.

2.1 BrbLib V6.00 – 2024-09-13

2.1.1 Portierung auf neuere Versionen

Beim Entwicklungs-Projekt wurden einige Versionen hochgezogen:

	Alte Version	Neue Version
Automation Studio	4.12.5.95	6.0.2.177
Compiler	6.3.0	11.3.0
Automation Runtime	14.93	6.0.2
VC4	4.73.1	6.0.0

2.1.2 Hinweise auf AS

In diese Hilfe wurden Hinweise zum Automation Studio aufgenommen. Darin steht u.a., dass die Bibliothek auch von **AS4.9 bis AS4.12 unter Compiler V6.3.0** eingesetzt werden kann.

Siehe [Hinweise zum AS und Compiler](#).

2.1.3 Erweiterung Schrittketten-Protokollierung um 2. Variante

Die Schrittketten-Protokollierung wurde um eine Variante erweitert, welche auch Zeitangaben pro Schritt protokolliert. So kann der Anwender entscheiden, welche Variante hilfreicher ist.

Siehe [StepHandling](#).

2.1.4 Neue Funktion BrbGetTimeOfDay

Die Funktion gibt die aktuelle Tageszeit als `TIME_OF_DAY` zurück.

3 Fehlernummern

In der Enumeration ‚BrbError_ENUM‘ sind Fehlernummern definiert, welche von Funktionen oder Funktionsblöcken im Fehlerfall zurückgegeben werden:

3.1 eBRB_ERR_OK = 0

Die Funktion wurde korrekt ausgeführt.

3.2 eBRB_ERR_NULL_POINTER = 50000

Ein Eingangs-Parameter, der als Pointer übergeben werden muss, wurde als 0 übergeben.

3.3 eBRB_ERR_INVALID_PARAMETER = 50001

Ein Eingangs-Parameter enthält einen ungültigen Wert (z.B. eine TextSize ist 0).

3.4 eBRB_ERR_NOT_ENABLED = 50002

Ein Funktionsblock wurde aufgerufen, obwohl dessen Enable-Eingang nicht gesetzt ist.

3.5 eBRB_ERR_OVERFLOW = 50003

Bei einer Rechenoperation trat ein Unter- oder Überlauf auf.

3.6 eBRB_ERR_DIVISION_BY_0 = 50004

Eine Division durch 0 ist nicht erlaubt.

3.7 eBRB_ERR_STRING_TOO_SHORT = 50005

Ein übergebener String ist zu kurz, um den Text aufzunehmen.

3.8 eBRB_ERR_STRING_TOO_LONG = 50006

Ein übergebener String ist zu lang für die Operation.

3.9 eBRB_ERR_BUSY = 65535

Der Funktionsblock benötigt noch einen Aufruf.

4 Pakete

4.1 StepHandling

In diesem Paket finden sich Funktionen für komfortable Schrittketten-Behandlung. Diese gibt es 2 Varianten. So kann der Anwender entscheiden, welche Variante hilfreicher ist.

4.1.1 Variante 1

Diese Variante verbraucht nicht so viel CPU-Leistung und Speicher, enthält aber weniger Informationen in der Schrittketten-Protokollierung als [Variante 2](#).

Zuerst müssen die Schritte in einer lokalen Enumeration „Steps_ENUM“ und die dargestellte Struktur lokal deklariert werden:

1	Steps_ENUM			
12	eSTEP_INIT			1
12	eSTEP_INIT_WAIT_FOR_PAR_VALID			
12	eSTEP_INIT_FINISHED			
12	eSTEP_WAIT_FOR_COMMAND			100
12	eSTEP_CMD1			
12	eSTEP_CMD1_FINISHED			
	TemplateStep_TYP			
	sStepText	STRING[nBRB_STEP_TEXT_CHAR_MAX]	<input type="checkbox"/>	
	eStepNr	Steps_ENUM	<input type="checkbox"/>	
	bInitDone	BOOL	<input type="checkbox"/>	

Weiterhin müssen die Variablen „Step“ und „StepHandling“ lokal angelegt werden:

Step	TemplateStep_TYP
StepHandling	BrbStepHandling_TYP

Hier die Beispiel-Schrittkette. Der Absatz mit dem Aufruf der Funktion „StepHandler“ muss unbedingt vorhanden sein:

```
// StepHandling
if (StepHandling.Current.bTimeoutElapsed == 1)
{
    StepHandling.Current.bTimeoutElapsed = 0;
    Step.eStepNr = StepHandling.Current.nTimeoutContinueStep;
}
StepHandling.Current.nStepNr = (DINT)Step.eStepNr;
strcpy(StepHandling.Current.sStepText, Step.sStepText);
BrbStepHandler(&StepHandling);

// Schrittkette
switch (Step.eStepNr)
{
    //-----
    case eSTEP_INIT:
        strcpy(Step.sStepText, "eSTEP_INIT");
        Step.eStepNr = eSTEP_INIT_FINISHED;
        break;

    case eSTEP_INIT_FINISHED:
        strcpy(Step.sStepText, "eSTEP_INIT_FINISHED");
        Step.bInitDone = 1;
        BrbClearCallerBox((UDINT)&gTemplate.CallerBox, sizeof(gTemplate.CallerBox));
        Step.eStepNr = eSTEP_WAIT_FOR_COMMAND;
        break;
```

```
//-----  
case eSTEP_WAIT_FOR_COMMAND:  
    strcpy(Step.sStepText, "eSTEP_WAIT_FOR_COMMAND");  
    if(gTemplate.CallerBox.bDummy == 1)  
    {  
        Step.eStepNr = eSTEP_CMD1;  
    }  
    break;  
  
//-----  
case eSTEP_CMD1:  
    strcpy(Step.sStepText, "eSTEP_CMD1");  
    // Beispiel für die Benutzung eines Flags  
    if(StepHandling.Flags.bInit == 0)  
    {  
        StepHandling.Flags.bInit = 1;  
        // ...  
    }  
    Step.eStepNr = eSTEP_CMD1_FINISHED;  
    break;  
  
case eSTEP_CMD1_FINISHED:  
    strcpy(Step.sStepText, "eSTEP_CMD1_FINISHED");  
    BrbClearCallerBox((UDINT)&gTemplate.CallerBox, sizeof(gTemplate.CallerBox));  
    Step.eStepNr = eSTEP_WAIT_FOR_COMMAND;  
    break;  
  
}
```

4.1.1.1 BrbStepHandler

`signed long` BrbStepHandler(`struct` BrbStepHandling_TYP* pStepHandling)

Argumente:

`struct` BrbStepHandling_TYP* pStepHandling
Zeiger auf eine Instanz von "BrbStepHandling_TYP"

Rückgabe:

`DINT`
eBRB_ERR_OK = 0

Beschreibung:

Die Funktion muss zyklisch aufgerufen werden (siehe Beispiel oben).

StepHandling	BrbStepHandling_TYP	local	
Current	BrbStepHandlingCurrent_TYP		
nStepNr	DINT		100
sStepText	STRING[50]		'eSTEP_WAIT_FOR_COMMAND'
bTimeoutElapsed	BOOL		FALSE
nTimeoutContinueStep	DINT		0
Flags	BrbStepHandlingCurrentFlags_TYP		
bInit	BOOL		FALSE
bUser	BOOL[0..7]		
bUser[0]	BOOL		FALSE
bUser[1]	BOOL		FALSE
bUser[2]	BOOL		FALSE
bUser[3]	BOOL		FALSE
bUser[4]	BOOL		FALSE
bUser[5]	BOOL		FALSE
bUser[6]	BOOL		FALSE
bUser[7]	BOOL		FALSE
Log	BrbStepHandlingLog_TYP		
bClear	BOOL		FALSE
bStop	BOOL		FALSE
Steps	BrbStepHandlingStep_TYP[0..20]		
Steps[0]	BrbStepHandlingStep_TYP		
nStepNr	DINT		100
sStepText	STRING[50]		'eSTEP_WAIT_FOR_COMMAND'
nCycleCount	UDINT		1
Steps[1]	BrbStepHandlingStep_TYP		
nStepNr	DINT		3
sStepText	STRING[50]		'eSTEP_INIT_FINISHED'
nCycleCount	UDINT		1
Steps[2]	BrbStepHandlingStep_TYP		
Steps[3]	BrbStepHandlingStep_TYP		
Steps[4]	BrbStepHandlingStep_TYP		

Folgende Funktionalitäten sind inkludiert:

-Flags für Schritt-Wechsel

Diese Flags können vom Anwender frei benutzt werden, um in jedem Schritt einmalig Aufgaben auszuführen.

Der Anwender muss dazu das jeweilige Flag auf `FALSE` abfragen, die Aufgabe ausführen und dann das Flag auf `TRUE` setzen. Bei einem Schrittwechsel werden alle Flags automatisch von der Bibliothek zurückgesetzt (siehe Schrittketten-Beispiel `eSTEP_CMD1` oben).

-Zeitüberwachung eines Schritts (Timeout oder Wartezeit; siehe unten)

-Schrittprotokollierung mit Nummer, Text und Zyklus-Dauer der letzten 21 Schritte in der Struktur „StepHandling“:

-Mit dem Kommando „bClear“ kann die Protokollierung gelöscht werden

-Mit dem Kommando „bStop“ kann die Protokollierung angehalten werden

4.1.1.2 BrbStartStepTimeout

```
unsigned short BrbStartStepTimeout(struct BrbStepHandling_TYP* pStepHandling, unsigned long nTimeout, signed long nContinueStep)
```

Argumente:

```
struct BrbStepHandling_TYP* pStepHandling
    Zeiger auf eine Instanz von "BrbStepHandling_TYP"
unsigned long nTimeout
    Zeitangabe in [ms]
signed long nContinueStep
    Schrittnummer bei abgelaufenem Timeout
```

Rückgabe:

```
UINT
```

```
eBRB_ERR_OK = 0
```

Beschreibung:

Starten einer Schritt-Zeitüberwachung. Wenn die Zeit abgelaufen ist, wird auf den angegebenen Schritt gewechselt.
Damit kann eine Wartezeit realisiert werden.

4.1.1.3 BrbStopStepTimeout

```
unsigned short BrbStopStepTimeout(struct BrbStepHandling_TYP* pStepHandling)
```

Argumente:

```
struct BrbStepHandling_TYP* pStepHandling  
    Zeiger auf eine Instanz von "BrbStepHandling_TYP"
```

Rückgabe:

```
UINT  
    eBRB_ERR_OK = 0
```

Beschreibung:

Stoppen einer Schritt-Zeitüberwachung vor Ablauf.
Damit kann z.B. das Überwachen eines Rückmelde-Signals realisiert werden.

4.1.2 Variante 2

Diese Variante verbraucht etwas mehr CPU-Leistung und Speicher als [Variante 1](#), enthält aber auch Zeitangaben über Eintritt und Austritt sowie die Dauer des Schritts in der Schrittketten-Protokollierung. Dazu müssen andere Versionen der Struktur-Typen und Funktionen verwendet werden.

Zuerst müssen wie bei Variante 1 die Schritte in einer lokalen Enumeration „Steps_ENUM“ und die dargestellte Struktur lokal deklariert werden:

Steps_ENUM			
eSTEP_INIT			1
eSTEP_INIT_WAIT_FOR_PAR_VALID			
eSTEP_INIT_FINISHED			
eSTEP_WAIT_FOR_COMMAND			100
eSTEP_CMD1			
eSTEP_CMD1_FINISHED			
TemplateStep_TYP			
sStepText	STRING[hBRB_STEP_TEXT_CHAR_MAX]	<input type="checkbox"/>	
eStepNr	Steps_ENUM	<input type="checkbox"/>	
blnitDone	BOOL	<input type="checkbox"/>	

Weiterhin müssen die Variablen „Step“ und „StepHandling“ lokal angelegt werden:

Step	TemplateStep_TYP	
StepHandling	BrbStepHandling2_TYP	

Hier die Beispiel-Schrittkette. Der Absatz mit dem Aufruf der Funktion „StepHandler2“ muss unbedingt vorhanden sein:

```
// StepHandling  
if (StepHandling.Current.bTimeoutElapsed == 1)  
{  
    StepHandling.Current.bTimeoutElapsed = 0;  
    Step.eStepNr = StepHandling.Current.nTimeoutContinueStep;  
}  
StepHandling.Current.nStepNr = (DINT) Step.eStepNr;  
strcpy (StepHandling.Current.sStepText, Step.sStepText);  
BrbStepHandler2 (&StepHandling);
```

```
// Schrittkette
switch (Step.eStepNr)
{
    //-----
    case eSTEP_INIT:
        strcpy(Step.sStepText, "eSTEP_INIT");
        Step.eStepNr = eSTEP_INIT_FINISHED;
        break;

    case eSTEP_INIT_FINISHED:
        strcpy(Step.sStepText, "eSTEP_INIT_FINISHED");
        Step.bInitDone = 1;
        BrbClearCallerBox((UDINT)&gTemplate.CallerBox, sizeof(gTemplate.CallerBox));
        Step.eStepNr = eSTEP_WAIT_FOR_COMMAND;
        break;

    //-----
    case eSTEP_WAIT_FOR_COMMAND:
        strcpy(Step.sStepText, "eSTEP_WAIT_FOR_COMMAND");
        if(gTemplate.CallerBox.bDummy == 1)
        {
            Step.eStepNr = eSTEP_CMD1;
        }
        break;

    //-----
    case eSTEP_CMD1:
        strcpy(Step.sStepText, "eSTEP_CMD1");
        // Beispiel für die Benutzung eines Flags
        if(StepHandling.Flags.bInit == 0)
        {
            StepHandling.Flags.bInit = 1;
            // ...
        }
        Step.eStepNr = eSTEP_CMD1_FINISHED;
        break;

    case eSTEP_CMD1_FINISHED:
        strcpy(Step.sStepText, "eSTEP_CMD1_FINISHED");
        BrbClearCallerBox((UDINT)&gTemplate.CallerBox, sizeof(gTemplate.CallerBox));
        Step.eStepNr = eSTEP_WAIT_FOR_COMMAND;
        break;
}
```

4.1.2.1 BrbStepHandler2

`signed long` BrbStepHandler2(`struct` BrbStepHandling2_TYP* pStepHandling)

Argumente:

`struct` BrbStepHandling2_TYP* pStepHandling
Zeiger auf eine Instanz von "BrbStepHandling2_TYP"

Rückgabe:

`DINT`
eBRB_ERR_OK = 0

Beschreibung:

Die Funktion muss zyklisch aufgerufen werden (siehe Beispiel oben).

StepHandling	BrbStepHandling2_TYP	local	
Current	BrbStepHandlingCurrent_TYP		
nStepNr	DINT		100
sStepText	STRING[50]		'eSTEP_WAIT_FOR_COMMAND'
bTimeoutElapsed	BOOL		FALSE
nTimeoutContinueStep	DINT		102
Flags	BrbStepHandlingCurrentFlags_TYP		
bInit	BOOL		FALSE
bUser	BOOL[0..7]		
bUser[0]	BOOL		FALSE
bUser[1]	BOOL		FALSE
bUser[2]	BOOL		FALSE
bUser[3]	BOOL		FALSE
bUser[4]	BOOL		FALSE
bUser[5]	BOOL		FALSE
bUser[6]	BOOL		FALSE
bUser[7]	BOOL		FALSE
Log	BrbStepHandlingLog2_TYP		
bClear	BOOL		FALSE
bStop	BOOL		FALSE
Steps	BrbStepHandlingStep2_TYP[0..20]		
Steps[0]	BrbStepHandlingStep2_TYP		
Steps[1]	BrbStepHandlingStep2_TYP		
nStepNr	DINT		103
sStepText	STRING[50]		'eSTEP_CMD1_FINISHED'
nCycleCount	UDINT		1
Time	BrbStepHandlingStepTime2_TYP		
todEnter	TIME_OF_DAY		TOD#11:08:00.526
todLeave	TIME_OF_DAY		TOD#11:08:00.526
tPeriod	TIME		T#0ms
Steps[2]	BrbStepHandlingStep2_TYP		
Steps[3]	BrbStepHandlingStep2_TYP		
Steps[4]	BrbStepHandlingStep2_TYP		

Es sind dieselben Funktionen inkludiert wie bei [Variante 1](#):

-Flags für Schritt-Wechsel

Diese Flags können vom Anwender frei benutzt werden, um in jedem Schritt einmalig Aufgaben auszuführen.

Der Anwender muss dazu das jeweilige Flag auf `FALSE` abfragen, die Aufgabe ausführen und dann das Flag auf `TRUE` setzen. Bei einem Schrittwechsel werden alle Flags automatisch von der Bibliothek zurückgesetzt (siehe Schrittketten-Beispiel `eSTEP_CMD1` oben).

-Zeitüberwachung eines Schritts (Timeout oder Wartezeit; siehe unten)

-Schrittprotokollierung mit Nummer, Text und Zyklus-Dauer der letzten 21 Schritte in der Struktur „StepHandling“:

-Mit dem Kommando „bClear“ kann die Protokollierung gelöscht werden

-Mit dem Kommando „bStop“ kann die Protokollierung angehalten werden

Zusätzlich sind folgende Informationen pro Schritt enthalten (siehe Screenshot oben):

-Eintritts-Zeitstempel `todEnter` als `TIME_OF_DAY`

-Austritts-Zeitstempel `todLeave` als `TIME_OF_DAY`

-Schritt-Dauer `tPeriod` als `TIME`

Hinweis: Da die Zeit-Erfassungen nicht zum Zeitpunkt der Schrittausführung, sondern in der Bibliothek stattfinden, sind sie nur annähernd exakt.

Hinweis: Bei der Berechnung von `tPeriod` wird der Übergang um Mitternacht aus Performance-Gründen nicht berücksichtigt. In diesem Fall wird eine falsche Dauer berechnet.

4.1.2.2 BrbStartStepTimeout2

```
unsigned short BrbStartStepTimeout2(struct BrbStepHandling2_TYP* pStepHandling, unsigned long nTimeout, signed long nContinueStep)
```

Argumente:

```
struct BrbStepHandling2_TYP* pStepHandling  
    Zeiger auf eine Instanz von "BrbStepHandling2_TYP"  
unsigned long nTimeout  
    Zeitangabe in [ms]  
signed long nContinueStep  
    Schrittnummer bei abgelaufenem Timeout
```

Rückgabe:

```
UINT  
eBRB_ERR_OK = 0
```

Beschreibung:

Starten einer Schritt-Zeitüberwachung. Wenn die Zeit abgelaufen ist, wird auf den angegebenen Schritt gewechselt.
Damit kann eine Wartezeit realisiert werden.

4.1.2.3 BrbStopStepTimeout2

```
unsigned short BrbStopStepTimeout2(struct BrbStepHandling2_TYP* pStepHandling)
```

Argumente:

```
struct BrbStepHandling2_TYP* pStepHandling  
    Zeiger auf eine Instanz von "BrbStepHandling2_TYP"
```

Rückgabe:

```
UINT  
eBRB_ERR_OK = 0
```





Beschreibung:

Stoppen einer Schritt-Zeitüberwachung vor Ablauf.
Damit kann z.B. das Überwachen eines Rückmelde-Signals realisiert werden.

4.1.3 Stoppuhr

Die Stoppuhr kann zum Messen von Abläufen verwendet werden.

4.1.3.1 BrbStartStopWatch

BrbStopWatch_TYP			
	tStartTime	TIME	<input type="checkbox"/> Start-Zeitstempel
	tStopTime	TIME	<input type="checkbox"/> End-Zeitstempel
	nTimeDiff	UDINT	<input type="checkbox"/> Berechnete Differenz
	sTimeDiff	STRING[24]	<input type="checkbox"/> Differenz als Text

```
plcbit BrbStartStopWatch(struct BrbStopWatch_TYP* pStopWatch)
```

Argumente:

```
struct BrbStopWatch_TYP* pStopWatch  
    Zeiger auf eine Instanz von "BrbStopWatch_TYP"
```

Rückgabe:

```
UINT  
eBRB_ERR_OK = 0
```

Beschreibung:

Starten einer Stoppuhr.

Damit kann eine Zeit von 1 Millisekunde bis 24 Tage gemessen werden.

4.1.3.2 BrbStopStopWatch

```
unsigned long BrbStopStopWatch(struct BrbStopWatch_TYP* pStopWatch)
```

Argumente:

```
struct BrbStopWatch_TYP* pStopWatch  
    Zeiger auf eine Instanz von "BrbStopWatch_TYP"
```

Rückgabe:

```
UDINT  
    Zeitmessung in [ms]
```

Beschreibung:

Stoppen einer Stoppuhr.

Die Struktur enthält die gemessene Zeit als UDINT in Millisekunden und als Text.

4.2 TaskCommunication

In diesem Paket finden sich Funktionen zum Kommunizieren im Taskklassen-System.

Es geht dabei um das Absetzen von Kommandos an einen Task, welches gewöhnlich durch eine BOOL-Variable realisiert wird.

Man unterscheidet DirectBox-Kommandos, welche in einem Zyklus ausgeführt werden können und CallerBox-Kommandos, welche mehrere Zyklen für die Ausführung brauchen (z.B. wenn es in einer Schrittkette abgearbeitet wird).

Bei CallerBox-Kommandos muss das erneute Setzen während der Ausführzeit verriegelt werden.

Normalerweise sind DirectBox- und CallerBox-Kommandos in zwei Unterstrukturen aufgeteilt:

gTemplate	Template_TYP	global
CallerBox	TemplateCallerBox_TYP	
Caller	BrbCaller_TYP	
nCallerId	DINT	
bLock	BOOL	
bDummy	BOOL	
DirectBox	TemplateDirectBox_TYP	
bDummy	BOOL	
Par	TemplatePar_TYP	
nDummy	UINT	
State	TemplateState_TYP	
nDummy	DINT	

Da die Tasks der höheren Taskklassen die Tasks der niedrigeren Taskklassen unterbrechen, muss das Verriegeln über eine Funktion erfolgen, welche diese Unterbrechungen berücksichtigt.

Vorraussetzung dafür ist eine Instanz der Struktur „BrbCaller_TYP“, die sich in der CallerBox-Struktur befindet sowie eine eindeutige Nummer für jeden Task. Der Identifier für jeden Task kann über eine globale Enumerierung definiert werden:

1 CallerIds_ENUM	
12 eCALLERID_NONE	
12 eCALLERID_TEMPLATE	
12 eCALLERID_PARHANDLING	
12 eCALLERID_VISU	

4.2.1 BrbSetCaller

BrbCallerStates_ENUM BrbSetCaller(struct BrbCaller_TYP* pCaller, signed long nCallerId)

Argumente:

struct BrbCaller_TYP* pCaller
Zeiger auf eine Instanz von "BrbCaller_TYP"
DINT nCallerId
Eindeutige Nummer des aufrufenden Tasks

Rückgabe:

BrbCallerStates_ENUM
Status als Enumeration:

1 BrbCallerStates_ENUM		
12 eBRB_CALLER_STATE_NOT_READY	-1	-1=Nicht bereit
12 eBRB_CALLER_STATE_OK	0	0=Bereit
12 eBRB_CALLER_STATE_BUSY	1	1=Besetzt

Beschreibung:

Versucht einen Task für die Ausführung eines Kommandos zu verriegeln.

Wenn der Status „eBRB_CALLER_STATE_OK“ lautet, ist der ausführende Task frei und das Kommando darf abgesetzt werden.

4.2.2 BrbClearDirectBox

```
unsigned short BrbClearDirectBox(unsigned long pDirectBox, unsigned long nSize)
```

Argumente:

UDINT pDirectBox
Zeiger auf die komplette DirectBox
UDINT nSize
Die Größe der Struktur

Rückgabe:

UINT
eBRB_ERR_OK = 0

Beschreibung:

Löscht die gesamte DirectBox. Sie darf nur vom ausführenden Task aufgerufen werden.

4.2.3 BrbClearCallerBox

```
unsigned short BrbClearCallerBox(unsigned long pCallerBox, unsigned long nSize)
```

Argumente:

UDINT pCallerBox
Zeiger auf die komplette CallerBox
UDINT nSize
Die Größe der Struktur

Rückgabe:

UINT
eBRB_ERR_OK = 0

Beschreibung:

Löscht die gesamte CallerBox mit der in ihr enthaltenen Struktur „BrbCaller_TYP“. Dadurch wird automatisch auch die Reservierung aufgehoben. Sie darf nur vom ausführenden Task aufgerufen werden.

4.3 VarHandling

In diesem Paket finden sich Funktionen für das Behandeln von Variablen, z.B. das Speichern und Laden von Variablen-Inhalten.

4.3.1 BrbSaveVarAscii

```
void BrbSaveVarAscii(struct BrbSaveVarAscii* inst)
```

Argumente:

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

Eingänge:

```
STRING* pDevice  
    Zeiger auf den Laufwerks-Namen  
STRING* pFile  
    Zeiger auf den Datei-Namen inkl. Pfad  
STRING* pVarName  
    Zeiger auf den Variablen-Namen  
UINT nLinesToWriteAtOneStep  
    Anzahl der Zeilen, die in einem Aufruf geschrieben werden
```

Ausgänge:

```
UDINT nCharCountMaxPerWrite  
    Größte Anzahl der Zeichen, welche in einer Zeile geschrieben wurden  
UINT nStatus  
    Funktionsblock-Status  
    eBRB_ERR_OK = 0  
    eBRB_ERR_NULL_POINTER = 50000  
    eBRB_ERR_BUSY = 65535  
    1..65534 = Fehlermeldung. Wenn es sich um einen System-Fehler (<50000) handelt, kann die Beschreibung dazu  
    in der Studio-Hilfe gefunden werden.
```

Beschreibung:

Schreibt den Inhalt einer Variablen als Ascii-Text in eine Datei.

Dieser Fub sollte nur aus einer Restzeit-Task aufgerufen werden, weil es sonst zu einer Zykluszeit-Verletzung kommen könnte.

Durch den Parameter „nLinesToWriteAtOneStep“ kann festgelegt werden, wie viele Zeilen in einem Zyklus geschrieben werden. Ein guter Standard-Wert dafür ist 100. Die Anzahl der in einem Zyklus geschriebenen Ascii-Zeichen darf nicht über 50000 liegen, da es sonst zu Fehlfunktionen führen kann.

Hinweis: Als Zeilenumbruch wird immer CRLF (CarriageReturn+LineFeed) verwendet.

Vorteile:

- Die Datei kann in einem Editor bearbeitet werden
- Alte Dateien können geladen werden, auch wenn die Variablen-Struktur geändert wurde

Nachteile:

- Nicht sehr schnell
- Es werden nur die gängigsten Datentypen unterstützt:
 - BOOL
 - SINT
 - INT
 - DINT
 - USINT
 - UINT
 - UDINT
 - REAL
 - STRING
 - DATE_AND_TIME
- Enthaltene Arrays dürfen nur bei Index 0 beginnen

Achtung: Enumerationen werden nicht unterstützt. Sie können aber in einem DINT-Item abgelegt werden.

Bei sehr großen Strukturen sollte aus zeitlichen Gründen besser die binäre Variante benutzt werden.
Achtung: Es gibt leider keine Funktion zum Umwandeln eines LREAL in STRING. Daher wird ein LREAL intern erst in einen REAL gewandelt. Dies hat leider einen Genauigkeits-Verlust zur Folge!

4.3.2 BrbLoadVarAscii

```
void BrbLoadVarAscii(struct BrbLoadVarAscii* inst)
```

Argumente:

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

Eingänge:

```
STRING* pDevice  
    Zeiger auf den Laufwerks-Namen  
STRING* pFile  
    Zeiger auf den Datei-Namen inkl. Pfad  
BrbLineBreak_ENUM eLineBreak  
    Verwendeter Zeilenumbruch
```

Verwendeter Zeilenumbruch	
BrbLineBreak_ENUM	Angabe des Zeilenumbruchs
eBRB_LINE_BREAK_CRLF	0=CarriageReturn+LineFeed
eBRB_LINE_BREAK_CR	1=CarriageReturn
eBRB_LINE_BREAK_LF	2=LineFeed

```
UINT nLinesToReadAtOneStep  
    Anzahl der Zeilen, die in einem Aufruf gelesen werden
```

Ausgänge:

```
UINT nStatus  
    Funktionsblock-Status  
    eBRB_ERR_OK = 0  
    eBRB_ERR_NULL_POINTER = 50000  
    eBRB_ERR_BUSY = 65535  
    1..65534 = Fehlermeldung. Wenn es sich um einen System-Fehler (<50000) handelt, kann die Beschreibung dazu  
    in der Studio-Hilfe gefunden werden.
```

Beschreibung:

Liest den Inhalt einer Variablen aus einer Ascii-Datei.

Damit auch extern erstellte Dateien eingelesen werden können, kann der darin verwendete Zeilenumbruch angegeben werden.

Dieser Fub sollte nur aus einer Restzeit-Task aufgerufen werden, weil es sonst zu einer Zykluszeit-Verletzung kommen könnte.

Durch den Parameter „nLinesToReadAtOneStep“ kann festgelegt werden, wie viele Zeilen in einem Zyklus gelesen werden. Ein guter Standard-Wert dafür ist 100.

Vorteile:

- Die Datei kann in einem Editor bearbeitet werden
- Alte Dateien können geladen werden, auch wenn die Variablen-Struktur geändert wurde

Nachteile:

- Nicht sehr schnell
- Es werden nur die gängigsten Datentypen unterstützt:
 - BOOL
 - SINT
 - INT
 - DINT
 - USINT
 - UINT
 - UDINT
 - REAL
 - STRING
 - DATE_AND_TIME

-Enthaltene Arrays dürfen nur bei Index 0 beginnen

Achtung: Enumerationen werden nicht unterstützt. Sie können aber in einem DINT-Item abgelegt werden.

Bei sehr großen Strukturen sollte aus zeitlichen Gründen besser die binäre Variante benutzt werden.

4.3.3 BrbSaveVarBin

```
void BrbSaveVarBin(struct BrbSaveVarBin* inst)
```

Argumente:

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

Eingänge:

```
STRING* pDevice  
    Zeiger auf den Laufwerks-Namen  
STRING* pFile  
    Zeiger auf den Datei-Namen inkl. Pfad  
STRING* pVarName  
    Zeiger auf den Variablen-Namen
```

Ausgänge:

```
UINT nStatus  
    Funktionsblock-Status  
    eBRB_ERR_OK = 0  
    eBRB_ERR_NULL_POINTER = 50000  
    eBRB_ERR_BUSY = 65535  
    1..65534 = Fehlermeldung. Wenn es sich um einen System-Fehler (<50000) handelt, kann die Beschreibung dazu  
    in der Studio-Hilfe gefunden werden.
```

Beschreibung:

Schreibt den Inhalt einer Variablen in eine Binär-Datei.

Vorteile:

- Auch bei großen Variablen sehr schnell

Nachteile:

- Keine Veränderungen an der Variablen-Struktur erlaubt

Bei sehr großen Strukturen sollte aus zeitlichen Gründen diese binäre Variante benutzt werden.

4.3.4 BrbLoadVarBin

```
void BrbLoadVarBin(struct BrbLoadVarBin* inst)
```

Argumente:

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

Eingänge:

```
STRING* pDevice  
    Zeiger auf den Laufwerks-Namen  
STRING* pFile  
    Zeiger auf den Datei-Namen inkl. Pfad  
STRING* pVarName  
    Zeiger auf den Variablen-Namen  
BOOL bAllowBiggerVar  
    Gibt an, ob der Speicher der Variablen größer als die Datei sein darf
```

Ausgänge:

```
UINT nStatus  
    Funktionsblock-Status  
    eBRB_ERR_OK = 0  
    eBRB_ERR_NULL_POINTER = 50000  
    eBRB_ERR_OVERFLOW = 50003 (Datei-Länge passt nicht zu Variablen-Länge)  
    eBRB_ERR_BUSY = 65535  
    1..65534 = Fehlermeldung. Wenn es sich um einen System-Fehler (<50000) handelt, kann die Beschreibung dazu  
    in der Studio-Hilfe gefunden werden.
```

Beschreibung:

Liest den Inhalt einer Variablen aus einer Binär-Datei.

Vorteile:

- Auch bei großen Variablen sehr schnell

Nachteile:

- Keine Veränderungen an der Variablen-Struktur erlaubt

Bei sehr großen Strukturen sollte aus zeitlichen Gründen diese binäre Variante benutzt werden.

Normalerweise wird die Größe der Datei mit der Größe der Variablen verglichen, welche gleich sein müssen. Wird eine Struktur nur verlängert, könnte die Datei also noch passen. Dann kann mit dem Eingang „bAllowBiggerVar“ erreicht werden, dass die Datei geladen wird.

4.4 FileHandling

In diesem Paket finden sich Funktionen für Laufwerks- und Datei-Behandlung.

4.4.1 BrbCheckUsbSticks

```
void BrbCheckUsbSticks(struct BrbCheckUsbSticks* inst)
```

Argumente:

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

Eingänge:

```
BOOL* bAutolink  
    Gibt an, ob ein erkannter Usb-Stick automatisch als Laufwerk gelinkt werden soll
```

Ausgänge:

```
UDINT nUsbDeviceCount  
    Anzahl der gesteckten Usb-Sticks  
BOOL bUsbDeviceCountChanged  
    Für einen Zyklus auf 1, wenn sich die Liste geändert hat  
BrbUsbDeviceListEntry_TYP[0..nBRB_USB_DEVICE_LIST_INDEX_MAX] UsbDeviceList  
    Eine Liste mit Informationen über die gesteckten Sticks  
UINT nStatus  
    Funktionsblock-Status  
    eBRB_ERR_OK = 0  
    eBRB_ERR_BUSY = 65535  
    1..65534 = Fehlermeldung. Wenn es sich um einen System-Fehler (<50000) handelt, kann die Beschreibung dazu  
    in der Studio-Hilfe gefunden werden.
```

Beschreibung:

Gibt Informationen über gesteckte Usb-Sticks zurück.

Dieser Funktionsblock sollte zyklisch aufgerufen werden. Das Stecken und Ziehen wird automatisch erkannt und der Stick dann optional auch als Laufwerk gelinkt und ungelinkt.

Die Ausgangsliste enthält die Informationen:

BrbUsbDeviceListEntry_TYP				
◆	sInterfaceName	STRING[nBRB_DEVICE_NAME_CHAR_MAX]	<input type="checkbox"/>	Schnittstellen-Name
◆	sDeviceName	STRING[nBRB_DEVICE_NAME_CHAR_MAX]	<input type="checkbox"/>	Schnittstellen-Name
◆	nNode	UDINT	<input type="checkbox"/>	Interne Node-Nummer
◆	nHandle	UDINT	<input type="checkbox"/>	Internes Handle

Ein Handle ist nur dann vorhanden, wenn das Device gelinkt wurde.

Hinweis: Es werden auch B&R-Dongles erkannt und gelinkt, da auch sie ein Speichermedium darstellen.

Achtung: Die Ausgangs-Liste darf nicht verändert werden!

4.4.2 BrbReadDir

```
void BrbReadDir(struct BrbReadDir* inst)
```

Argumente:

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

Eingänge:

```
STRING* pDevice  
    Zeiger auf den Laufwerks-Namen  
STRING* pPath  
    Zeiger auf einen optionalen Pfad. Wenn nicht benötigt, dann 0  
BrbDirInfoFilter eFilter  
    Angabe zur Filterung als Enumeration:
```

BrbDirInfoFilter_ENUM		
eBRB_DIR_INFO_ONLY_FILES		0=Nur Dateien
eBRB_DIR_INFO_ONLY_DIRS		1=Nur Verzeichnisse
eBRB_DIR_INFO_FILES_AND_DIRS		2=Dateien und Verzeichnisse

BOOL bWithParentDir

Gibt an, ob das übergeordnete Verzeichnis „..“ mitgeliefert wird

STRING* pFileFilter

Zeiger auf den Text mit den Filterangaben (Dateiendungen getrennt durch „;“, z.B. „txt;html+xml“)

BOOL bUserFilter

Gibt an, ob eine zusätzliche Filterung durch eine benutzerdefinierte Filterfunktion ausgeführt werden soll

UDINT pUserFilterFunction

Zeiger auf eine benutzerdefinierte Filterfunktion. Wenn nicht benötigt, dann 0

BrbFileSorting_ENUM eSorting

Angabe zur Sortierung als Enumeration:

BrbFileSorting_ENUM		
eBRB_FILE_SORTING_NONE		0=Keine Sortierung
eBRB_FILE_SORTING_ALPH_UP		1=Sortierung nach aufsteigendem Alphabet
eBRB_FILE_SORTING_ALPH_DOWN		2=Sortierung nach aufsteigendem Alphabet
eBRB_FILE_SORTING_OLDEST		3=Sortierung nach ältesten Dateien
eBRB_FILE_SORTING_YOUNGEST		4=Sortierung nach jüngsten Dateien
eBRB_FILE_SORTING_BIGGEST		5=Sortierung nach größten Dateien
eBRB_FILE_SORTING_SMALLEST		6=Sortierung nach kleinsten Dateien
eBRB_FILE_SORTING_USER		7=Sortierung mit benutzerdefinierter Vergleichsfunktion

BOOL bCaseSensitive

Der Parameter wird nur bei alphanumerischer Sortierung berücksichtigt. Er gibt an, ob die Einträge mit Berücksichtigung der Groß-/Kleinschreibung sortiert werden. Ist er 0, werden Kleinbuchstaben wie Großbuchstaben behandelt.

UDINT pUserCompareFunction

Zeiger auf eine benutzerdefinierte Vergleichsfunktion. Wenn nicht benötigt, dann 0

UDINT pList

Zeiger auf ein Array vom Typ „BrbReadDirListEntry_TYP“

UDINT nListIndexMax

Größter zulässiger Index des Arrays

Ausgänge:

UDINT nDirCount

Anzahl der Verzeichnisse, die der Filterung entsprechen

UDINT nFileCount

Anzahl der Dateien, die der Filterung entsprechen

UDINT nTotalCount

Anzahl der Verzeichnisse und Dateien, die der Filterung entsprechen

UINT nStatus

Funktionsblock-Status

eBRB_ERR_OK = 0

eBRB_ERR_NULL_POINTER = 50000

eBRB_ERR_BUSY = 65535

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

20729 = Benutzerdefinierte Filterfunktion bzw. Sortierfunktion nicht angegeben

Beschreibung:

Füllt eine Liste mit den Informationen der Dateien eines Laufwerks.

Dazu muss ein Array vom Typ „BrbReadDirListEntry_TYP“ instanziiert werden, welches dann vom Fub gefüllt wird. Die Filterung und Sortierung kann dem Fub übergeben werden.

BrbReadDirListEntry_TYP			
sName	STRING[nBRB_FILE_NAME_CHAR_MAX]	<input type="checkbox"/>	Verzeichnis- oder Datei-Name
dtDate	DATE_AND_TIME	<input type="checkbox"/>	Zeitstempel
nSize	UDINT	<input type="checkbox"/>	Größe (0=Verzeichnis)

Achtung: Die Anzahl-Ausgänge zeigen immer die tatsächliche Anzahl der Elemente, auch wenn die Liste nicht groß genug ist, um sie aufzunehmen.

Hinweis zur Performance: Zur Überprüfung des Filters muss natürlich jedes Unterverzeichnis und jede Datei gelesen werden. Wenn die übergebene Liste nicht groß genug ist, um alle gefilterten Elemente aufzunehmen, muss die Liste nach jedem Einlesen eines neuen Elements sortiert werden, um

den letzten Eintrag mit dem neuen zu vergleichen. Nur so kann entschieden werden, ob der neue Eintrag laut Sortierung den alten ersetzen muss. Diese temporären Sortierungen können entfallen, wenn die Liste groß genug gewählt wird. Damit wird natürlich auch die Performance gesteigert.

Filterung durch benutzerdefinierte Funktion:

Hinweis: Diese Funktionalität ist aufgrund von Funktionszeigern nur in ANSI-C nutzbar, aber nicht in IEC-Sprachen (siehe Punkt 1.1 [Hinweise zu StructuredText und anderen IEC-Sprachen](#))

Ist der Eingang ‚bUserFilter‘ auf 1, so wird am Eingang ‚pUserFilterFunction‘ die Adresse einer Filterfunktion erwartet. Ist dies nicht der Fall, wird der Status ‚fiERR_PARAMETER‘ (=20729) zurückgegeben.

Der Name der Filterfunktion kann selbst definiert werden, sie **muss** jedoch folgende Signatur haben:

```
BOOL ReadDirUserFilterFunction(BrbReadDirListEntry_TYP* pEntry)
```

Das bedeutet, dass es einen Parameter mit dem Zeiger-Datentyp ‚BrbReadDirListEntry_TYP*‘ geben und der Rückgabewert vom Datentyp ‚BOOL‘ sein muss.

Sie kann wie folgt am Eingang gesetzt werden:

```
fbBrbReadDir.pUserFilterFunction = (UDINT)&ReadDirUserFilterFunction;
```

Diese Filterfunktion wird dann während des Einlesens für jeden Eintrag aufgerufen.

Innerhalb der Funktion kann dann auf die Elemente des Eintrags zugegriffen werden und entschieden werden, ob die Filterkriterien erfüllt sind.

Über den Rückgabewert wird festgelegt, ob der Eintrag in die Liste übernommen werden soll:

0	=	Eintrag wird in die Liste übernommen
1	=	Eintrag wird nicht in die Liste übernommen

Wichtig: Dies ist eine zusätzliche Filterung. Die Funktion wird nur für Einträge aufgerufen, welche die ursprüngliche Filterung durch die Eingänge ‚eFilter‘ und ‚pFileFilter‘ überstanden haben. Der Eintrag „..“ für das übergeordnete Verzeichnis wird nicht der Funktion übergeben, sondern nur durch den Eingang ‚bWithParentDir‘ gefiltert.

Sortierung ‚eBRB_FILE_SORTING_USER‘:

Hinweis: Diese Funktionalität ist aufgrund von Funktionszeigern nur in ANSI-C nutzbar, aber nicht in IEC-Sprachen (siehe Punkt 1.1 [Hinweise zu StructuredText und anderen IEC-Sprachen](#))

Bei dieser Einstellung muss der Eingang ‚pUserCompareFunction‘ mit der Adresse einer Vergleichsfunktion besetzt werden. Wenn dies nicht geschieht, wird der Status ‚fiERR_PARAMETER‘ (=20729) zurückgegeben.

Der Name der Vergleichsfunktion kann selbst definiert werden, sie **muss** jedoch folgende Signatur haben:

```
INT ReadDirUserCompareFunction(BrbReadDirListEntry_TYP* pEntry1, BrbReadDirListEntry_TYP* pEntry2)
```

Das bedeutet, dass es zwei Parameter mit dem Zeiger-Datentyp ‚BrbReadDirListEntry_TYP*‘ geben und der Rückgabewert vom Datentyp ‚INT‘ sein muss.

Sie kann wie folgt am Eingang gesetzt werden:

```
fbBrbReadDir.pUserCompareFunction = (UDINT)&ReadDirUserCompareFunction;
```

Diese Vergleichsfunktion wird dann während des Sortierens mehrmals mit unterschiedlichen Einträgen aufgerufen.

Innerhalb der Funktion kann dann auf die Elemente beider Einträge zugegriffen werden und somit ein Vergleich stattfinden.

Über den Rückgabewert wird festgelegt, wie Eintrag#1 gegenüber Eintrag#2 einsortiert werden soll:

-1	=	Eintrag#1 oberhalb von Eintrag#2
0	=	Eintrag#1 und Eintrag#2 sind gleichwertig
1	=	Eintrag#1 unterhalb von Eintrag#2

Durch diese Methodik kann die Liste nach komplett selbst zu definierenden Kriterien sortiert werden.

4.4.3 BrbDeleteFiles

```
void BrbDeleteFiles(struct BrbDeleteFiles* inst)
```

Argumente:

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

Eingänge:

`STRING*` pDevice
Zeiger auf den Laufwerks-Namen
`STRING*` pPath
Zeiger auf einen optionalen Pfad. Wenn nicht benötigt, dann 0
`STRING*` pFileFilter
Zeiger auf den Text mit den Filterangaben (Dateiendungen getrennt durch „;“, z.B. „txt;html+xml“)
`DATE_AND_TIME` dtStartDate
Start-Datum für Zeitraum
`DATE_AND_TIME` dtStartEnd
Ende-Datum für Zeitraum

Ausgänge:

`UDINT` nDeletedFileCount
Anzahl der gelöschten Dateien
`UDINT` nKeptFileCount
Anzahl der nicht gelöschten Dateien
`UINT` nStatus
Funktionsblock-Status
eBRB_ERR_OK = 0
eBRB_ERR_NULL_POINTER = 50000
eBRB_ERR_BUSY = 65535
1..65534 = Fehlermeldung. Wenn es sich um einen System-Fehler (<50000) handelt, kann die Beschreibung dazu in der Studio-Hilfe gefunden werden.

Beschreibung:

Löscht Dateien aus einem Verzeichnis, welche bestimmte Bedingungen erfüllen.

Dazu gehören Datei-Endungen sowie ein Zeitraum, der über Start- und Ende-Zeitstempel (inklusive) vorgegeben wird.

4.4.4 BrbLoadFileDataObj

```
void BrbLoadFileDataObj(struct BrbLoadFileDataObj* inst)
```

Argumente:

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

Eingänge:

`STRING*` pDevice
Zeiger auf den Laufwerks-Namen
`STRING*` pFile
Zeiger auf den Datei-Namen inkl. Pfad
`STRING*` pDataObjName
Zeiger auf den Datenobjekt-Namen
`UDINT` nDataObjMemType
Speicher, in dem das Datenobjekt erzeugt werden soll (Konstanten aus der Bibliothek „DataObj“):
doSYSROM = 0
doUSRROM = 2
doUSRRAM = 3
doMEMCARD = 4
doFIXRAM = 5
doTEMP = 65
`UDINT` nDataObjMemOption
Option für das Erzeugen des Datenobjekt (Konstanten aus der Bibliothek „DataObj“):
doNO_CS = 1

Ausgänge:

`UINT` nStatus
Funktionsblock-Status
eBRB_ERR_OK = 0
eBRB_ERR_NULL_POINTER = 50000
eBRB_ERR_BUSY = 65535
1..65534 = Fehlermeldung. Wenn es sich um einen System-Fehler (<50000) handelt, kann die Beschreibung dazu in der Studio-Hilfe gefunden werden.
`UDINT` nDataObjIdent
Ident des erzeugten Datenobjekts
`UDINT` nDataObjMem

Zeiger auf die Daten des Datenobjekts
`UDINT nDataObjLen`
Länge des erzeugten Datenobjekts (entspricht der Dateilänge)

Beschreibung:

Lädt eine Datei in ein Datenobjekt. Ein evtl. schon existierendes Datenobjekt dieses Namens wird vorher gelöscht. Tritt ein Fehler auf, wird die Datei automatisch geschlossen und das Datenobjekt gelöscht.

Wichtig: Tritt kein Fehler auf, sollte das Datenobjekt applikativ gelöscht werden, wenn es nicht mehr benötigt wird.

Die anzugebenden Konstanten sind in der System-Bibliothek „DataObj“ nachzulesen.

4.4.5 BrbSaveFileDataObj

```
void BrbSaveFileDataObj(struct BrbSaveFileDataObj* inst)
```

Argumente:

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

Eingänge:

`STRING* pDevice`
Zeiger auf den Laufwerks-Namen
`STRING* pFile`
Zeiger auf den Datei-Namen inkl. Pfad
`STRING* pDataObjName`
Zeiger auf den Datenobjekt-Namen

Ausgänge:

`UINT nStatus`
Funktionsblock-Status
`eBRB_ERR_OK = 0`
`eBRB_ERR_NULL_POINTER = 50000`
`eBRB_ERR_BUSY = 65535`
1..65534 = Fehlermeldung. Wenn es sich um einen System-Fehler (<50000) handelt, kann die Beschreibung dazu in der Studio-Hilfe gefunden werden.

Beschreibung:

Speichert ein Datenobjekt in einer Datei. Eine evtl. schon existierende Datei dieses Namens wird vorher gelöscht.

4.4.6 BrbLoadFileBin

```
void BrbLoadFileBin(struct BrbLoadFileBin* inst)
```

Argumente:

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

Eingänge:

`STRING* pDevice`
Zeiger auf den Laufwerks-Namen
`STRING* pFile`
Zeiger auf den Datei-Namen inkl. Pfad
`STRING* pVar`
Zeiger auf die Feld-Variable vom Typ USINT
`UDINT nVarSize`
Größe der angegebenen Feld-Variable (muss mindestens der Datei-Größe entsprechen)

Ausgänge:

`UINT nStatus`
Funktionsblock-Status
`eBRB_ERR_OK = 0`
`eBRB_ERR_NULL_POINTER = 50000`
`eBRB_ERR_BUSY = 65535`
1..65534 = Fehlermeldung. Wenn es sich um einen System-Fehler (<50000) handelt, kann die Beschreibung dazu in der Studio-Hilfe gefunden werden.
`UDINT nValidBytes`

Anzahl der gelesenen Bytes (entspricht der Dateigröße)

Beschreibung:

Liest eine Datei in eine Feld-Variable vom Typ USINT.

4.4.7 BrbCheckFileName

```
unsigned short BrbCheckFileName(plcstring* pFileName)
```

Argumente:

```
plcstring* pFileName  
    Zeiger auf den Dateinamen
```

Rückgabe:

```
UINT  
    Anzahl der ersetzten Zeichen  
nBRB_ERR_NULL_POINTER
```

Beschreibung:

Diese Funktion ersetzt alle Zeichen außer diesen mit Unterstrichen:

- „0“ bis „1“
- „a“ bis „z“
- „A“ bis „Z“
- „-“
- der letzte Punkt (zur Trennung der Datei-Erweiterung)

Damit kann ein eingegebener Text zur Verwendung als Dateiname umgewandelt werden.

4.4.8 BrbCheckFileEnding

```
plcbit BrbCheckFileEnding(plcstring* pFileName, plcstring* pEnding)
```

Argumente:

```
plcstring* pFileName  
    Zeiger auf den Dateinamen  
plcstring* pEnding  
    Zeiger auf die Dateiendung
```

Rückgabe:

```
BOOL  
    0=Fehler (Null-Pointer)  
    1=Endung wurde angepasst
```

Beschreibung:

Diese Funktion sorgt dafür, dass die übergebene Endung am Dateinamen angehängt ist. Dabei spielt es keine Rolle, ob schon eine Endung vorhanden ist und ob sie einen Punkt enthält.

4.4.9 BrbCombinePath

```
unsigned short BrbCombinePath(plcstring* pPath, plcstring* pFilename, plcstring* pFilenameWithPath)
```

Argumente:

```
plcstring* pPath  
    Zeiger auf den Pfad  
plcstring* pFilename  
    Zeiger auf einen Pfad oder Dateinamen  
plcstring* pFilenameWithPath  
    Zeiger auf den String, der das Ergebnis aufnimmt
```

Rückgabe:

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

Beschreibung:

Diese Funktion fügt zwei Pfade oder einen Pfad und einen Dateinamen zusammen. Das Trennzeichen „\“ wird wenn nötig eingefügt.

4.5 Logger

In diesem Paket finden sich Funktionen zur Behandlung von Logbüchern und deren Einträgen.

4.5.1 BrbLoggerReadHierarchicalList

```
void BrbLoggerReadHierarchicalList(struct BrbLoggerReadHierarchicalList* inst)
```

Argumente:

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

Eingänge:

`STRING[256] sLogbookName`

Name des Logbuchs. Siehe AS-Hilfe für FB „ArEventLogGetIdent“

`ArEventLogRecordIDType nStartRecordId`

Datentyp entspricht UDINT. RecordId, bei dem die Suche beginnen soll (0=Beginn bei jüngstem Eintrag)

`STRING[36] sObjectId`

Filter der ObjectId (entspricht der Spalte „Entered by“, „“ = keine Filterung)

`DINT nStartEventId`

EventId oder ErrorNumber, bei dem die Liste beginnen soll (0=keine Filterung)

`BrbLogHierarchListSevFilter_ENUM eSeverities`

Bitweise codierter Filter der Severity. Die einzelnen Angaben können addiert werden:

BrbLogHierarchListSevFilter_ENUM			
		eBRB_LOG_SEVERITY_FILTER_ALL	0 Keine Filterung
		eBRB_LOG_SEVERITY_FILTER_SUCCESS	1 Nur Erfolgs-Einträge
		eBRB_LOG_SEVERITY_FILTER_INFO	2 Nur Info-Einträge
		eBRB_LOG_SEVERITY_FILTER_WARNING	4 Nur Warnungs-Einträge
		eBRB_LOG_SEVERITY_FILTER_ERROR	8 Nur Fehler-Einträge

`ArEventLogLanguageCodeType sLanguageCode`

Datentyp entspricht STRING[18]. Gibt als Sprach-Kürzel an, in welcher Sprache die Beschreibung ermittelt wird, z.B. „de“ oder „en“. Siehe AS-Hilfe für FB „ArEventLogReadDescription“

`UDINT nSearchEntryCountLimit`

Limitierung der durchsuchten Einträge (0=kein Limit)

`UDINT pLogList`

Zeiger auf ein Array vom Typ „BrbLogHierarchListEntry_TYP“

`UINT nLogListIndexMax`

Größter zulässiger Index des Arrays

Ausgänge:

`UINT nListEntryCount`

Anzahl der gültigen Einträge

`DINT nStatus`

Funktionsblock-Status

eBRB_ERR_OK = 0

eBRB_ERR_BUSY = 65535

xxx = Fehlermeldung. Wenn es sich um einen System-Fehler handelt, kann die Beschreibung dazu in der Studio-Hilfe gefunden werden.

Beschreibung:

Liest die zu einem definierten Logger-Eintrag gehörende Hierarchie-Liste von Einträgen aus.

Begleitende Hinweise:

In AS4.2 wurde das Logbuch-Konzept der AR überarbeitet. Seitdem gibt es zwei Arten eines Logbucheintrags (siehe AS-Hilfe):

1. Alter Eintrag im ErrorNumber-Format

Wird von älteren Bibliotheken verwendet.

Die als EventId angezeigte ErrorNumber ist ein UDINT und enthält die Fehlernummer des alten Formats. Die Severity (Info, Warnung, Fehler) ist extra abgelegt.

Ein Verweis auf einen anderen Eintrag ist nicht möglich.

2. Neuer Eintrag im EventId-Format

Wird von neueren Bibliotheken verwendet, insbesondere von mapp-Bibliotheken.

Die EventId ist ein teilweise bitcodierter DINT. Die Severity (Success, Info, Warnung, Fehler) ist in den Bits 30..31 abgelegt.

Der Eintrag kann einen Verweis auf einen ursächlichen Eintrag haben. Daraus kann sich eine hierarchische Verkettung von Einträgen ergeben.


Hinweis: Der Funktionsblock kann mit beiden Eintrags-Formaten arbeiten.

Jeder Eintrag hat eine sogenannte RecordId, die eine eindeutige Nummer im gesamten Logbuch darstellt. Sie ist normalerweise aufsteigend in der Reihenfolge der Eintragung.

Der Funktionsblock liefert nun die Liste an verketteten Einträgen. Der Start-Eintrag kann über Filter bestimmt werden.

Die Liste muss applikativ angelegt und dem FB als Zeiger übergeben werden. Das bietet den Vorteil, dass der Speicherverbrauch vom Benutzer beeinflusst werden kann.

Dazu muss ein Array vom Typ „BrbLogHierarchListEntry_TYP“ instanziiert werden, welches dann vom Fub gefüllt wird.



BrbLogHierarchListEntry_TYP	
nRecordId	ArEventLogRecordIDType
nOriginRecordId	ArEventLogRecordIDType
nEventId	DINT
nSeverity	USINT
sObjectId	STRING[36]
sDescription	STRING[512]

nRecordId	Eindeutige Id des Eintrags innerhalb des Logbuchs
nOriginRecordId	Verweis auf einen übergeordneten Eintrag
nEventId	EventId oder ErrorNumber
nSeverity	0=Success, 1=Info, 2=Warnung oder 3=Fehler
sObjectId	Angabe des Teilnehmers, der den Eintrag erstellt hat
sDescription	Sprachübersetzter Text

Arbeitsweise:

Der Name des Logbuchs muss übergeben werden, z.B. ‚\$arlogsys‘ oder ‚\$motion‘. Siehe dazu die AS-Hilfe für den FB ‚ArEventLogGetIdent‘.

Ist eine ‚nStartRecordId‘ angegeben, wird dieser Eintrag gelesen und ab diesem die Suche begonnen. Ist der Eintrag nicht vorhanden, wird mit einer Fehlermeldung geendet.

Ist keine ‚nStartRecordId‘ (=0) angegeben, so wird die Suche beim jüngsten Eintrag begonnen.

Hier beginnt die Filterung in dieser Reihenfolge:

Filterung ‚sObjectId‘

Wenn keine ‚sObjectId‘ (="") angegeben ist, wird nicht gefiltert.

Ist eine ‚sObjectId‘ angegeben und stimmt mit dem aktuellen Eintrag überein, wird zur nächsten Stufe gesprungen. Stimmt sie nicht überein und ist noch gar kein Eintrag gefunden, wird der nächst jüngste Eintrag ermittelt. Ansonsten wird der FB beendet.

Da z.B. bei mappMotion diese Spalte den Namen des MpLinks enthält, kann nach einem Fehler dieser Achse gesucht werden.

Filterung ‚nStartEventId‘

Wenn keine ‚nStartEventId‘ (=0) angegeben ist, wird nicht gefiltert.

Ist eine ‚nStartEventId‘ angegeben und stimmt mit dem aktuellen Eintrag überein, wird zur nächsten Stufe gesprungen. Stimmt sie nicht überein und ist noch gar kein Eintrag gefunden, wird der nächst jüngste Eintrag ermittelt. Ansonsten wird der FB beendet.

Ist es ein alter Eintrag, wird die ErrorNumber als EventId verwendet.

Liefert z.B. ein mappMotion-FB einen Fehler, kann dieser hier als Filter übergeben werden.

So wird genau dieser Fehler-Eintrag gesucht.

Filterung ‚eSeverities‘

Die einzelnen Enum-Angaben können addiert werden.

Wenn ‚eSeverities‘ auf ‚eBRB_LOG_SEVERITY_FILTER_ALL‘ steht, wird nicht gefiltert.

Stimmt der aktuelle Eintrag mit dem Filter überein, wird zur nächsten Stufe gesprungen.

Stimmt sie nicht überein und ist noch gar kein Eintrag gefunden, wird der nächst jüngste Eintrag ermittelt. Ansonsten wird der FB beendet.

Durch den bitweise codierten Filter werden nur Einträge dieser Severities (Success, Info, Warnung und/oder Fehler) in die Liste übernommen. Manchmal werden bei Fehlern als Info eingetragene Einträge als Ursache angegeben (z.B. bei mappMotion-Achsfehlern, wenn ein Bewegungskommando einen Fehler verursacht hat). Durch die Filterung auf ‚eBRB_LOG_SEVERITY_FILTER_ERROR‘ werden dann die Info-Einträge nicht in die Liste eingetragen.

Hier wird dann die Beschreibung des Eintrags (also der Text) gelesen und der Eintrag in die Liste übernommen. Ist ein Verweis auf einen verketteten Eintrag vorhanden, wird dieser gelesen und die Filterung beginnt von vorn. Ist kein Verweis vorhanden, wird der FB beendet.

Solange kein geeigneter Eintrag lt. Filter gefunden wird, wird der chronologisch nächste Eintrag ausgelesen und geprüft. Wenn nun der Filter so gesetzt ist, dass kein Eintrag passt, werden also ALLE Einträge gelesen und geprüft.

Wenn ein Logbuch sehr viele Einträge (z.B. > 10000) hat, kann die Suche also sehr lange dauern, bevor eine leere Liste zurückgegeben wird. Für diesen Fall kann die Suche durch ‚nSearchEntryCountLimit‘ auf eine Anzahl von Einträgen beschränkt werden. Ist nach dem Durchsuchen dieser Anzahl von Einträgen immer noch kein passender gefunden, wird der FB beendet. Ist dieser Eingang 0, gibt es keine Limitierung und es wird tatsächlich das ganze Logbuch durchsucht.

Die Liste enthält nach Beendigung des FB die ermittelten Einträge, wobei der älteste (also der ursächlichste) Eintrag ganz unten steht.

Beispiel: Ermitteln des Eintrags eines mappMotion-Achsfehlers und dessen Ursachen

sLogbook	=	„\$motion“
nStartRecordId	=	0
sObjectId	=	Name des MappLinks, z.B. „mplAxRvManiMain“
nStartEventId	=	StatusID des Motion-Funktionsblocks , z.B. MpAxisBasic.Info.Diag.Internal.ID = -1067317248
eSeverities	=	eBRB_LOG_SEVERITY_FILTER_ERROR
sLanguageCode	=	“en”
nSearchEntryCountLimit	=	200

Der FB ermittelt den ersten Eintrag der Achse mit der angegebenen Fehlernummer und trägt ihn in der Liste ein. Dann ermittelt er die ursächlichen Einträge und übernimmt diese ebenfalls in die Liste. Hier ein Beispiel der Ergebnis-Liste:

LogErrorList	BrbLogHierarchListEntry_TYP	
LogErrorList[0]	BrbLogHierarchListEntry_TYP	
nRecordId	ArEventLogRecordIDType	70
nOriginRecordId	ArEventLogRecordIDType	69
nEventId	DINT	-1067317247
nSeverity	USINT	3
sObjectId	STRING[36]	'mplAxRvManiMain'
sDescription	STRING[512]	'Target position is outside the axis period'
LogErrorList[1]	BrbLogHierarchListEntry_TYP	
nRecordId	ArEventLogRecordIDType	71
nOriginRecordId	ArEventLogRecordIDType	70
nEventId	DINT	-1067317248
nSeverity	USINT	3
sObjectId	STRING[36]	'mplAxRvManiMain'
sDescription	STRING[512]	'Command: MoveAbsolute failed'
LogErrorList[2]	BrbLogHierarchListEntry_TYP	
nRecordId	ArEventLogRecordIDType	0
nOriginRecordId	ArEventLogRecordIDType	0

4.6 TimeAndDate

In diesem Paket finden sich Funktionen zur Behandlung von Zeit- und Datums-Angaben.

4.6.1 BrbSetDtStruct

```
plcdt BrbSetDtStruct(struct DTStructure* pDtStruct, unsigned short nYear, unsigned char nMonth,
unsigned char nDay, unsigned char nHour, unsigned char nMinute, unsigned char nSecond, unsigned
short nMillisecond, unsigned short nMicrosecond)
```

Argumente:

```
struct DTStructure* pDtStruct
    Zeiger auf die zu setzende Instanz
UINT nYear
    Jahr (1970..2106)
USINT nMonth
    Monat (1..12)
USINT nDay
    Tag (1..31)
USINT nHour
    Stunde (0..23)
USINT nMinute
    Minute (0..59)
USINT nSecond
    Sekunde (0..59)
UINT nMillisecond
    Millisekunde (0..999)
UINT nMicrosecond
    Mikrosekunde (0..999)
```

Rückgabe:

```
DATE_AND_TIME
    Die Zeit als DATE_AND_TIME (ohne Milli- und Mikrosekunden)
```

Beschreibung:

Setzt eine DTStructure-Instanz laut den Angaben.

Enthält mindestens ein Element eine falsche Angabe, so wird sowohl die Instanz als auch der Rückgabewert auf die höchste darstellbare DATE_AND_TIME gesetzt (0xFFFFFFFF = 4294967295 = 2106-02-07 06:28:15).

4.6.2 BrbSetDt

```
plcdt BrbSetDt(unsigned short nYear, unsigned char nMonth, unsigned char nDay, unsigned char
nHour, unsigned char nMinute, unsigned char nSecond)
```

Argumente:

```
UINT nYear
    Jahr (1970..2106)
USINT nMonth
    Monat (1..12)
USINT nDay
    Tag (1..31)
USINT nHour
    Stunde (0..23)
USINT nMinute
    Minute (0..59)
USINT nSecond
    Sekunde (0..59)
```

Rückgabe:

```
DATE_AND_TIME
    Die Zeit als DATE_AND_TIME
```

Beschreibung:

Gibt eine DATE_AND_TIME-Zeit laut den Angaben zurück.

Enthält mindestens ein Element eine falsche Angabe, so wird der Rückgabewert auf die höchste darstellbare DATE_AND_TIME gesetzt (0xFFFFFFFF = 4294967295 = 2106-02-07 06:28:15).

4.6.3 BrbSetTimespan

```
plctime BrbSetTimespan(signed char nSign, unsigned long nDays, unsigned long nHours, unsigned long nMinutes, unsigned long nSeconds, unsigned long nMilliseconds)
```

Argumente:

SINT	nSign	
	<0	-1 = Negative Zeitspanne
	>=0	+1 = Positive Zeitspanne
UDINT	nDays	Tage
UDINT	nHours	Stunden
UDINT	nMinutes	Minuten
UDINT	nSeconds	Sekunden
UDINT	nMilliseconds	Millisekunden

Rückgabe:

TIME
Zeitspanne

Beschreibung:

Gibt eine TIME-Zeitspanne laut den Angaben zurück.

Systemintern wird eine Zeitspanne als Millisekunden in einem vorzeichenbehafteter 32-Bit-Wert gehalten (also als DINT).

Eine Angabe darf auch mehr als seine natürliche Begrenzung enthalten. Z.B. darf der Sekunden-Wert nicht nur 59, sondern auch 3600 (=1 Stunde) enthalten.

Wichtig ist nur, dass insgesamt der Wertebereich des Datentyp TIME nicht unter- oder überschritten wird. Wertebereich:

TIME		Dezimal
-24d 20h 31m 23s 648ms	=	-2147483648
+24d 20h 31m 23s 647ms	=	+2147483647

Unterschreitet die Summe aller Angaben den Min-Wert, so wird der Min-Wert zurückgegeben.

Überschreitet die Summe aller Angaben den Max-Wert, so wird der Max-Wert zurückgegeben.

4.6.4 BrbSetTimespanT

```
plctime BrbSetTimespanT(plcstring* pTimeText)
```

Argumente:

STRING* pTimeText
Zeiger auf den String, der die Zeitangabe enthält

Rückgabe:

TIME
Zeitspanne

Beschreibung:

Setzt eine TIME-Zeitspanne laut Text-Einzel-Angaben. Dies kann als Ersatz für die unter IEC-Sprachen übliche, aber unter ANSI-C nicht vorhandene Syntax „T#“ (T#1d2h3m4s5ms) verwendet werden.

Die Zeitangabe wird als Text übergeben. Das Präfix „T#“ ist optional.

Die Angabe kann folgende Postfixes beinhalten:

d	Tage
h	Stunden
m	Minuten
s	Sekunden
ms	Millisekunden

Jedes Postfix ist optional, darf aber nur einmal vorkommen. Die Zahl vor dem Postfix gibt die Wertigkeit des Elements an. Ist keine Zahl vor einem Postfix angegeben, wird sie als 0 gewertet.

Am Beginn kann eine negative Zeitspanne angegeben werden („T#-„ oder „-„). Minuszeichen danach werden nicht gewertet.

Nicht definierte Zeichen werden überlesen und nicht gewertet (z.B. Leerzeichen oder Unterstriche).

Beispiele:

```
T#2d3h3m1s987ms
-2d 3h 3m 1s 987ms
T#-1d4m
3600s
+34ms
-67m
```

Systemintern wird eine Zeitspanne als Millisekunden in einem vorzeichenbehafteter 32-Bit-Wert gehalten (also als INT).

Eine Angabe darf auch mehr als seine natürliche Begrenzung enthalten. Z.B. darf der Sekunden-Wert nicht nur 59, sondern auch 3600 (=1 Stunde) enthalten.

Wichtig ist nur, dass insgesamt der Wertebereich des Datentyp TIME nicht unter- oder überschritten wird. Wertebereich:

TIME		Dezimal
-24d 20h 31m 23s 648ms	=	-2147483648
+24d 20h 31m 23s 647ms	=	+2147483647

Unterschreitet die Summe aller Angaben den Min-Wert, so wird der Min-Wert zurückgegeben.

Überschreitet die Summe aller Angaben den Max-Wert, so wird der Max-Wert zurückgegeben.

4.6.5 BrbGetTimeText

```
unsigned short BrbGetTimeText(struct RTctime_typ* pTime, plcstring* pText, unsigned long
nTextSize, plcstring* pFormat)
```

Argumente:

```
struct RTctime_typ* pTime
    Zeiger auf eine Zeitangabe
STRING* pText
    Zeiger auf den String, der gefüllt werden soll
UDINT nTextSize
    Größe des Strings, der gefüllt werden soll
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“
```

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.

4.6.6 BrbGetCurrentTimeText

```
unsigned short BrbGetCurrentTimeText(plcstring* pText, unsigned long nTextSize, plcstring* pFormat)
```

Argumente:

STRING* pText
Zeiger auf den String, der gefüllt werden soll

UDINT nTextSize
Größe des Strings, der gefüllt werden soll

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“

Rückgabe:

UINT

eBRB_ERR_OK = 0
eBRB_ERR_NULL_POINTER = 50000
eBRB_ERR_INVALID_PARAMETER = 50001

Beschreibung:

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

4.6.7 BrbGetTimeTextDtStruct

```
unsigned short BrbGetTimeTextDtStruct(struct DTStructure* pTime, plcstring* pText, unsigned long nTextSize, plcstring* pFormat)
```

Argumente:

struct DTStructure* pTime
Zeiger auf eine Zeitangabe

STRING* pText
Zeiger auf den String, der gefüllt werden soll

UDINT nTextSize
Größe des Strings, der gefüllt werden soll

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“

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.

4.6.8 BrbGetTimeTextDt

```
unsigned short BrbGetTimeTextDt(plcdt dtTime, plcstring* pText, unsigned long nTextSize, plcstring* pFormat)
```

Argumente:

<code>DATE_AND_TIME</code>	<code>dtTime</code>	Zeitangabe
<code>STRING*</code>	<code>pText</code>	Zeiger auf den String, der gefüllt werden soll
<code>UDINT</code>	<code>nTextSize</code>	Größe des Strings, der gefüllt werden soll
<code>STRING*</code>	<code>pFormat</code>	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“

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. Werte, die nicht in der Struktur enthalten sind (z.B. Millisekunden), werden auf 0 gesetzt.

4.6.9 BrbGetDtFromTimeText

```
plcdt BrbGetDtFromTimeText(plcstring* pTimeText, plcstring* pFormat)
```

Argumente:

<code>plcstring*</code>	<code>pTimeText</code>	Zeit-Text im übergebenen Format
<code>STRING*</code>	<code>pFormat</code>	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“

Rückgabe:

```
DATE_AND_TIME  
Gewandelte Zeit  
0, wenn Fehler (Null-Pointer)
```

Beschreibung:

Wandelt einen Text in eine Zeit um. Es ist darauf zu achten, dass die Schlüsselwörter im Format-Text an derselben Stelle wie im Text stehen, sonst werden u.U. falsche Werte zurückgegeben.

4.6.10 BrbRtcTimeToDtStruct

```
unsigned short BrbRtcTimeToDtStruct(struct RTctime_typ* pRtcTime, struct DTStructure* pDtStruct)
```

Argumente:

<code>struct</code>	<code>RTctime_typ* pRtcTime</code>	Zeiger auf eine Instanz von "RTctime_typ"
<code>struct</code>	<code>DTStructure* pDtStruct</code>	Zeiger auf eine Instanz von "DTStructure"

Rückgabe:

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

Beschreibung:

Wandelt eine Zeit im Rtc-Format in eine Struktur vom Typ „DTStructure“.

4.6.11 BrbDtStructCompare

```
plcbit BrbDtStructCompare(struct DTStructure* pDtStruct1, enum BrbTimeAndDateCompare_ENUM eCompare, struct DTStructure* pDtStruct2)
```

Argumente:

`struct` DTStructure* pDtStruct1
 Zeiger auf eine Zeitangabe
`struct` DTStructure* pDtStruct2
 Zeiger auf eine Zeitangabe
`enum` BrbTimeAndDateCompare_ENUM eCompare
 Vergleichsangabe als Enumeration
`eBRB_TAD_COMPARE_YOUNGER`
`eBRB_TAD_COMPARE_YOUNGEREQUAL`
`eBRB_TAD_COMPARE_EQUAL`
`eBRB_TAD_COMPARE_OLDEREQUAL`
`eBRB_TAD_COMPARE_OLDER`

Rückgabe:

`BOOL`
 Ergebnis des Vergleichs (1=positiv)
 0, wenn Fehler (Null-Pointer)

Beschreibung:

Vergleicht zwei DTStructures.

4.6.12 BrbDtStructAddDays

```
plcdt BrbDtStructAddDays(struct DTStructure* pDtStruct, signed long nDays)
```

Argumente:

`struct` DTStructure* pDtStruct
 Zeiger auf eine Zeitangabe
`DINT` nDays
 Angabe der zu addierenden Tage

Rückgabe:

`DATE_AND_TIME`
 Ergebnis als DATE_AND_TIME
 0, wenn Fehler (Null-Pointer)

Beschreibung:

Addiert die angegebenen Tage zu einer DTStructure. Ist diese Angabe negativ, so wird subtrahiert.

4.6.13 BrbDtStructAddHours

```
plcdt BrbDtStructAddHours(struct DTStructure* pDtStruct, signed long nHours)
```

Argumente:

`struct` DTStructure* pDtStruct
 Zeiger auf eine Zeitangabe
`DINT` nHours
 Angabe der zu addierenden Stunden

Rückgabe:

`DATE_AND_TIME`
 Ergebnis als DATE_AND_TIME
 0, wenn Fehler (Null-Pointer)

Beschreibung:

Addiert die angegebenen Stunden zu einer DTStructure. Ist diese Angabe negativ, so wird subtrahiert.

Wenn die zu addierende Stunden-Anzahl für einen DINT zu groß ist, kann durch vorherigen Aufruf von ‚BrbDtStructAddDays‘ die zu übergebende Anzahl erheblich verkleinert werden.

4.6.14 BrbDtStructAddMinutes

```
plcdt BrbDtStructAddMinutes(struct DTStructure* pDtStruct, signed long nMinutes)
```

Argumente:

`struct` DTStructure* pDtStruct
Zeiger auf eine Zeitangabe
`DINT` nMinutes
Angabe der zu addierenden Minuten

Rückgabe:

`DATE_AND_TIME`
Ergebnis als DATE_AND_TIME
0, wenn Fehler (Null-Pointer)

Beschreibung:

Addiert die angegebenen Minuten zu einer DTStructure. Ist diese Angabe negativ, so wird subtrahiert.

Wenn die zu addierende Minuten-Anzahl für einen DINT zu groß ist, kann durch vorherigen Aufruf von ‚BrbDtStructAddHours‘ die zu übergebende Anzahl erheblich verkleinert werden.

4.6.15 BrbDtStructAddSeconds

```
plcdt BrbDtStructAddSeconds(struct DTStructure* pDtStruct, signed long nSeconds)
```

Argumente:

`struct` DTStructure* pDtStruct
Zeiger auf eine Zeitangabe
`DINT` nSeconds
Angabe der zu addierenden Sekunden

Rückgabe:

`DATE_AND_TIME`
Ergebnis als DATE_AND_TIME
0, wenn Fehler (Null-Pointer)

Beschreibung:

Addiert die angegebenen Sekunden zu einer DTStructure. Ist diese Angabe negativ, so wird subtrahiert.

Wenn die zu addierende Sekunden-Anzahl für einen DINT zu groß ist, kann durch vorherigen Aufruf von ‚BrbDtStructAddMinutes‘ die zu übergebende Anzahl erheblich verkleinert werden.

4.6.16 BrbDtStructAddMilliseconds

```
plcdt BrbDtStructAddMilliseconds(struct DTStructure* pDtStruct, signed long nMilliseconds)
```

Argumente:

`struct` DTStructure* pDtStruct
Zeiger auf eine Zeitangabe
`DINT` nMilliseconds
Angabe der zu addierenden Millisekunden

Rückgabe:

`DATE_AND_TIME`
Ergebnis als DATE_AND_TIME
0, wenn Fehler (Null-Pointer)

Beschreibung:

Addiert die angegebenen Millisekunden zu einer DTStructure. Ist diese Angabe negativ, so wird subtrahiert.

Wenn die zu addierende Millisekunden-Anzahl für einen DINT zu groß ist, kann auch die Funktion ‚BrbDtStructAddMillisecondsLReal‘ (siehe unten) verwendet werden.

4.6.17 BrbDtStructAddMillisecondsLReal

```
plcdt BrbDtStructAddMillisecondsLReal(struct DTStructure* pDtStruct, double rMilliseconds)
```

Argumente:

`struct` DTStructure* pDtStruct
Zeiger auf eine Zeitangabe
`LREAL` rMilliseconds
Angabe der zu addierenden Millisekunden

Rückgabe:

`DATE_AND_TIME`
Ergebnis als DATE_AND_TIME
0, wenn Fehler (Null-Pointer)

Beschreibung:

Addiert die angegebenen Millisekunden zu einer DTStructure. Ist diese Angabe negativ, so wird subtrahiert.

4.6.18 BrbGetTimeTextMs

```
unsigned short BrbGetTimeTextMs(signed long nMilliseconds, plcstring* pText, unsigned long nTextSize, plcstring* pFormat, plcbit bClip)
```

Argumente:

`DINT` nMilliseconds
Anzahl der Millisekunden
`STRING*` pText
Zeiger auf den String, der gefüllt werden soll
`UDINT` nTextSize
Größe des Strings, der gefüllt werden soll
`STRING*` pFormat
Zeiger auf den String, der die Formatierung enthält
Tag „dd“ oder „d“
Stunde „hh“ oder „h“
Minute „MM“ oder „M“
Sekunde „ss“ oder „s“
Millisekunde „mil“
`BOOL` bClip
Abziehen der nicht im Format berücksichtigten Teilwerte

Rückgabe:

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

Beschreibung:

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

Durch den Eingang „bClip“ kann festgelegt werden, wie mit nicht im Format berücksichtigten Teilwerten umgegangen wird.

Beispiel:

```
nMilliseconds = 7384000 = 2 Stunden, 3 Minuten und 4 Sekunden  
Format= „hh:MM:ss“  
bClip = 0 oder 1  
Text = „02:03:04“        -> Alle Teilwerte in Format berücksichtigt
```

```
nMilliseconds = 7384000 = 2 Stunden, 3 Minuten und 4 Sekunden  
Format= „MM:ss“  
bClip = 0  
Text = „123:04“        -> Stunden werden als Minuten berechnet
```

```
nMilliseconds = 7384000 = 2 Stunden, 3 Minuten und 4 Sekunden
Format= „MM:ss“
bClip = 1
Text = „03:04“          -> Stunden werden nicht berechnet
```

4.6.19 BrbGetTimeDiffMsDtStruct

```
signed long BrbGetTimeDiffMsDtStruct(struct DTStructure* pTimeStampBase, struct DTStructure*
pTimeStampCompare)
```

Argumente:

```
struct DTStructure* pTimeStampBase
    Zeiger auf den Basis-Zeitstempel
struct DTStructure* pTimeStampCompare
    Zeiger auf den Vergleichs-Zeitstempel
```

Rückgabe:

DINT
Zeitdifferenz in Millisekunden
0, wenn Fehler (Null-Pointer)

Beschreibung:

Ermittelt die Zeit-Differenz zwischen zwei DtStruct-Zeiten in [ms].
Liegt der Vergleichs-Zeitstempel nach dem Basis-Zeitstempel, wird eine positive Differenz zurückgegeben.
Liegt der der Vergleichs-Zeitstempel vor dem Basis-Zeitstempel, wird eine negative Differenz zurückgegeben.

4.6.20 BrbGetWeekdayDtStruct

```
BrbWeekdays_ENUM BrbGetWeekdayDtStruct(struct DTStructure* pDtStruct)
```

Argumente:

```
struct DTStructure* pDtStruct
    Zeiger auf eine Zeitangabe
```

Rückgabe:

BrbWeekdays_ENUM
Angabe des Wochentags

BrbWeekdays_ENUM	Wochentage
eBRB_WD_SUNDAY	0=Sonntag
eBRB_WD_MONDAY	1=Montag
eBRB_WD_TUESDAY	2=Dienstag
eBRB_WD_WEDNESDAY	3=Mittwoch
eBRB_WD_THURSDAY	4=Donnerstag
eBRB_WD_FRIDAY	5=Freitag
eBRB_WD_SATURDAY	6=Samstag

Beschreibung:

Ermittelt den Wochentag einer DTStructure-Zeit.
Der Wert wird sowohl in die Struktur eingetragen als auch zurückgegeben.
Die Enumeration entspricht der Kodierung wie in der Definition des Datentyps 'DTStructure' angegeben (0=Sonntag, 6=Samstag).

4.6.21 BrbGetTimeOfDay

```
plctod BrbGetTimeOfDay(void)
```

Argumente:

Rückgabe:

TIME_OF_DAY
Ergebnis als TIME_OF_DAY

Beschreibung:

Gibt die aktuelle Tageszeit als `TIME_OF_DAY` zurück. Intern wird dazu die `RTC-Time` verwendet.

4.6.22 BrbGetWeekdayDt

```
BrbWeekdays_ENUM BrbGetWeekdayDt(plcDt dtTime)
```

Argumente:

`DATE_AND_TIME` dtTime
Zeitangabe

Rückgabe:

`BrbWeekdays_ENUM`

Angabe des Wochentags

BrbWeekdays_ENUM	Wochentage
eBRB_WD_SUNDAY	0=Sonntag
eBRB_WD_MONDAY	1=Montag
eBRB_WD_TUESDAY	2=Dienstag
eBRB_WD_WEDNESDAY	3=Mittwoch
eBRB_WD_THURSDAY	4=Donnerstag
eBRB_WD_FRIDAY	5=Freitag
eBRB_WD_SATURDAY	6=Samstag

Beschreibung:

Ermittelt den Wochentag einer `DATE_AND_TIME`-Zeit.

Die Enumeration entspricht der Kodierung wie in der Definition des Datentyps `'DTStructure'` angegeben (0=Sonntag, 6=Samstag).

4.6.23 BrbTimerSwitch

```
void BrbTimerSwitch(struct BrbTimerSwitch* inst)
```

Argumente:

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

BrbTimerSwitch				Zeitschaltuhr für einen Kanal
bEnable	BOOL	<input type="checkbox"/> VAR_INPUT		1=Baustein wird ausgeführt
bCmdSwitchOff	BOOL	<input type="checkbox"/> VAR_INPUT		1=Ausgangs-Kanal wird ausgeschaltet
bCmdSwitchOn	BOOL	<input type="checkbox"/> VAR_INPUT		1=Ausgangs-Kanal wird eingeschaltet
bCmdToggle	BOOL	<input type="checkbox"/> VAR_INPUT		1=Ausgangs-Kanal wird umgeschaltet
pUserTime	DTStructure	<input checked="" type="checkbox"/> VAR_INPUT		Optionaler Zeiger auf Benutzer-Zeitstempel
Parameter	BrbTimerSwitchPar_TYP	<input type="checkbox"/> VAR_INPUT		Parameter-Struktur
bOut	BOOL	<input type="checkbox"/> VAR_OUTPUT		Ausgangs-Kanal
dtsUsedTime	DTStructure	<input type="checkbox"/> VAR_OUTPUT		Verwendete Zeit
nSwitchCount	UDINT	<input type="checkbox"/> VAR_OUTPUT		Anzahl der Schaltungen

Beschreibung:

Dieser Funktionsblock stellt eine Zeitschaltuhr für einen Kanal zur Verfügung.

Er wird nur ausgeführt, wenn der Eingang `„bEnable“` 1 ist.

Mit den Kommandos `„bCmdSwitchOff“`, `„bCmdSwitchOn“` und `„bCmdToggle“` kann der Ausgangs-Kanal manuell aus-, ein- oder umgeschaltet werden.

Wenn der Eingangs-Zeiger `„pUserTime“` 0 ist, wird die System-Zeit intern ermittelt und verwendet.

Zeigt er auf eine Struktur vom Typ `„DTStructure“`, dann wird diese Zeit verwendet. Somit kann auch eine andere Zeit als die Systemzeit als Basis verwendet werden.

Das kann auch zur Optimierung der Performance benutzt werden. Wenn mehrere Instanzen dieses Funktionsblocks verwendet werden, kann die Systemzeit applikativ nur einmal ermittelt und dann jeder Instanz über diesen Eingang zur Verfügung gestellt werden.

Für Zeitangaben wird grundsätzlich die System-Struktur `„DTStructure“` verwendet:

DTStructure		date structure
year	UINT	year
month	USINT	month (1-12)
day	USINT	day (1-31)
wday	USINT	day of the week (0-6), e.g. 0 = Sunday, 6 = Saturday
hour	USINT	hours (0-23)
minute	USINT	minutes (0-59)
second	USINT	seconds (0-59)
millisec	USINT	milliseconds (0-999)
microsec	USINT	microseconds (0-999)

Die Items „millisec“ und „microsec“ werden hier nicht verwendet.

Für die Angabe des Wochentages gibt es eine Auflistung:

BrbTimerSwitchWeekdays_ENUM	Wochentage
eBRB_TIMERSWITCH_WD_SUNDAY	0=Sonntag
eBRB_TIMERSWITCH_WD_MONDAY	1=Montag
eBRB_TIMERSWITCH_WD_TUESDAY	2=Dienstag
eBRB_TIMERSWITCH_WD_WEDNESDAY	3=Mittwoch
eBRB_TIMERSWITCH_WD_THURSDAY	4=Donnerstag
eBRB_TIMERSWITCH_WD_FRIDAY	5=Freitag
eBRB_TIMERSWITCH_WD_SATURDAY	6=Samstag

Die Struktur „Parameter“ enthält die Schaltzeitpunkte:

BrbTimerSwitchPar_TYP			Parameter der Zeitschaltuhr
eMode	BrbTimerSwitchMode_ENUM		Modus der Schaltung (Sekunde, Minute, Stunde...)
TimePoint	BrbTimerSwitchParTimePoint_TYP[0..nBRB_TIMERSWIT...		Liste der Schaltpunkte

Durch den Eingang „eMode“ wird festgelegt, welche Werte der aktuellen Zeit verglichen werden, um eine Schaltung zu erkennen:

BrbTimerSwitchMode_ENUM	Modus der Schaltung
eBRB_TIMERSWITCH_MODE_MONTH	0=Monat
eBRB_TIMERSWITCH_MODE_WEEK	1=Woche
eBRB_TIMERSWITCH_MODE_DAY	2=Tag
eBRB_TIMERSWITCH_MODE_HOUR	3=Stunde
eBRB_TIMERSWITCH_MODE_MINUTE	4=Minute
eBRB_TIMERSWITCH_MODE_SECOND	5=Sekunde

Bei „eBRB_TIMERSWITCH_MODE_SECOND“ z.B. wird jedesmal eine Schaltung ausgelöst, wenn der aktuelle Sekunden-Wert dem eines Zeitschalt-Punktes entspricht.

Bei „eBRB_TIMERSWITCH_MODE_MINUTE“ wird jedesmal eine Schaltung ausgelöst, wenn der Sekunden-Wert und der Minuten-Wert dem eines Zeitschalt-Punktes entspricht.

Bei „eBRB_TIMERSWITCH_MODE_HOUR“ wird jedesmal eine Schaltung ausgelöst, wenn der Sekunden-Wert, der Minuten-Wert und der Stunden-Wert dem eines Zeitschalt-Punktes entspricht.

Bei „eBRB_TIMERSWITCH_MODE_DAY“ wird jedesmal eine Schaltung ausgelöst, wenn der Sekunden-Wert, der Minuten-Wert, der Stunden-Wert und der Tages-Wert dem eines Zeitschalt-Punktes entspricht.

Bei „eBRB_TIMERSWITCH_MODE_WEEK“ wird jedesmal eine Schaltung ausgelöst, wenn der Sekunden-Wert, der Minuten-Wert, der Stunden-Wert und der Wochentag-Wert dem eines Zeitschalt-Punktes entspricht.

Bei „eBRB_TIMERSWITCH_MODE_MONTH“ wird jedesmal eine Schaltung ausgelöst, wenn der Sekunden-Wert, der Minuten-Wert, der Stunden-Wert, der Tages-Wert und der Monats-Wert dem eines Zeitschalt-Punktes entspricht.

Es können bis zu 16 Zeitschaltpunkte festgelegt werden:

BrbTimerSwitchParTimePoint_TYP		Schaltpunkt
bActive	BOOL	<input type="checkbox"/> 1=Zeitpunkt ist aktiv
dtsTimePoint	DTStructure	<input type="checkbox"/> Zeitpunkt
eSwitchType	BrbTimerSwitchType_ENUM	<input type="checkbox"/> Typ der Schaltung (Aus, An, Umschalten, Puls)

Ist „bActive“ 0, wird der Zeitschaltpunkt nicht ausgewertet.

Die Struktur „dtsTimePoint“ enthält den Zeitschalt-Punkt. Angaben, welche lt. eingestelltem Modus nicht zur Auswertung verwendet werden (siehe oben), müssen auch nicht belegt werden. Der Wert „wday“ z.B. wird nur beim Modus „eBRB_TIMERSWITCH_MODE_WEEK“ ausgewertet.

Der Parameter „eSwitchType“ legt fest, welche Schaltung beim Erkennen des Zeit-Punktes ausgeführt wird:

BrbTimerSwitchType_ENUM	Typ der Schaltung
eBRB_TIMERSWITCH_TYPE_OFF	0 = Aus
eBRB_TIMERSWITCH_TYPE_ON	1 = An
eBRB_TIMERSWITCH_TYPE_TOGGLE	2 = Umschalten
eBRB_TIMERSWITCH_TYPE_IMPULSE	3 = Puls

Die Einstellung „eBRB_TIMERSWITCH_TYPE_IMPULSE“ bewirkt, dass der Ausgang genau für einen Zyklus auf 1 gesetzt wird.

Generell wird beim Erkennen eines Zeitschalt-Punktes nur einmal geschaltet. Sind mehrere exakt gleiche Zeitschalt-Punkte angegeben, wird nur der erste in der Liste ausgeführt.

Der Funktionsblock besitzt noch diverse Ausgänge:

bOut	BOOL	VAR_OUTPUT	Ausgangs-Kanal
dtsUsedTime	DTStructure	VAR_OUTPUT	Verwendete Zeit
nSwitchCount	UDINT	VAR_OUTPUT	Anzahl der Schaltungen

Der Ausgang „bOut“ ist der geschaltete Ausgangs-Kanal.

Der Ausgang „dtsUsedTime“ enthält die tatsächlich verwendete Zeit (siehe oben).

Der Ausgang „nSwitchCount“ zählt die von der Zeitschaltuhr ausgeführten Schaltungen. Manuelle Schaltungen werden nicht gezählt.

4.7 Strings + WcStrings

In diesem Paket finden sich Funktionen für komfortable String-Behandlung, auch für WcStrings. WcStrings („Wide character strings“ oder auch „Unicode strings“) besitzen 2 Byte pro Zeichen und werden durch den Standard-Datentypen „WSTRING“ unterstützt, welcher eigentlich ein UINT-Feld ist.

Da der Aufruf jeweils derselbe ist, wird jede doppelte Funktion nur für ASCII-Strings beschrieben.

4.7.1 BrbWcCopyStringtoWString

```
unsigned short BrbWcCopyStringtoWString(plcwstring* pWcString, plcstring* pString)
```

Argumente:

`WSTRING*` pWcString
Zeiger auf den WcString
`STRING*` pString
Zeiger auf den String

Rückgabe:

`UINT`
eBRB_ERR_OK = 0

Beschreibung:

Kopiert einen String auf einen Unicode-String.
Hinweis: Die High-Bytes aller Zeichen werden auf 0 gesetzt.

4.7.2 BrbWcCopyWStringtoString

```
unsigned short BrbWcCopyWStringtoString(plcstring* pString, plcwstring* pWcString)
```

Argumente:

`STRING*` pString
Zeiger auf den String
`WSTRING*` pWcString
Zeiger auf den WcString

Rückgabe:

`UINT`
eBRB_ERR_OK = 0

Beschreibung:

Kopiert einen Unicode-String auf einen String.
Achtung: Die High-Bytes aller Zeichen gehen dabei verloren!

4.7.3 BrbUsintToHex

```
unsigned short BrbUsintToHex(unsigned char nValue, plcstring* pHex, unsigned long nHexSize, plcbit bWithPraefix)
```

Argumente:

`USINT` nValue
Wert
`STRING*` pHex
Zeiger auf den String, der gefüllt werden soll
`UDINT` nHexSize
Größe des Strings, der gefüllt werden soll
`BOOL` bWithPraefix
1= Präfix „0x“ wird vorangestellt

Rückgabe:

`UINT`
eBRB_ERR_OK = 0

Beschreibung:

Wandelt einen USINT-Wert in eine hexadezimale Zeichenfolge mit dem optionalen Präfix „0x“.
Hinweis: Die komplementäre Funktion kann mit `BrbHexToUdint` (siehe unten) erfolgen.

4.7.4 BrbUsintArrayToHex

```
unsigned short BrbUsintArrayToHex(unsigned char* pArray, signed long nArrayLength, plcstring*
pHex, unsigned long nHexSize, plcbit bDescending)
```

Argumente:

USINT* pArray
Zeiger auf das Array

DINT nArrayLength
Länge des Arrays

STRING* pHex
Zeiger auf den String, der gefüllt werden soll

UDINT nHexSize
Größe des Strings, der gefüllt werden soll

BOOL bDescending
0= Aufsteigend, 1= Absteigend

Rückgabe:

UINT

eBRB_ERR_OK = 0
eBRB_ERR_NULL_POINTER = 50000
eBRB_ERR_INVALID_PARAMETER = 50001

Beschreibung:

Wandelt ein USINT-Array in eine hexadezimale Zeichenfolge.
Die Richtung kann mit bDescending festgelegt werden:
0 = Aufsteigend: Das Byte an Index#0 steht am Beginn des Hex-Strings.
1 = Absteigend: Das Byte an Index#0 steht am Ende des Hex-Strings.

4.7.5 BrbHexToUsintArray

```
unsigned short BrbHexToUsintArray(plcstring* pHex, unsigned char* pArray, signed long nArray-
Length, plcbit bDescending)
```

Argumente:

STRING* pHex
Zeiger auf den Hex-String

USINT* pArray
Zeiger auf das Array, das gefüllt werden soll

DINT nArrayLength
Länge des Arrays

BOOL bDescending
0= Aufsteigend, 1= Absteigend

Rückgabe:

UINT

eBRB_ERR_OK = 0
eBRB_ERR_NULL_POINTER = 50000
eBRB_ERR_INVALID_PARAMETER = 50001

Beschreibung:

Wandelt eine hexadezimale Zeichenfolge in ein USINT-Array. Es werden Groß- und Kleinbuchstaben akzeptiert. Normalerweise entsprechen 2 Hex-Zeichen genau 1 Byte (FF = 255). Wird eine ungerade Anzahl von Hex-Zeichen übergeben, so wird intern dem ersten Hex-Zeichen eine „0“ vorangestellt.
Die Richtung kann mit bDescending festgelegt werden:
0 = Aufsteigend: Der erste Hex-Wert steht am Beginn des Arrays.
1 = Absteigend: Der erste Hex-Wert steht am Ende des Arrays.

4.7.6 BrbUdintToAscii

```
unsigned short BrbUdintToAscii(unsigned long nValue, plcstring* pText)
```

Argumente:

UDINT nValue
Wert

STRING* pText
Zeiger auf den String, der gefüllt werden soll

Rückgabe:

UDINT

eBRB_ERR_OK = 0

Beschreibung:

Wandelt einen UDINT-Wert in eine Zeichenfolge. Das normal verwendete „itoa()“ nimmt nur DINT.

4.7.7 BrbAsciiToUdint

```
unsigned long BrbAsciiToUdint(plcstring* pText)
```

Argumente:

STRING* pText

Zeiger auf den String, der die Zahl enthält

Rückgabe:

UDINT

Ergebnis

Beschreibung:

Wandelt eine Zeichenfolge in einen UDINT-Wert. Das normal verwendete „atoi()“ nimmt nur DINT. Ist der im Text angegebene Wert zu groß für UDINT, wird der maximale Wert (4294967295) zurückgegeben.

4.7.8 BrbUdintToHex

```
unsigned short BrbUdintToHex(unsigned long nValue, plcstring* pHex, unsigned long nHexSize, plcbit bWithPraefix)
```

Argumente:

UDINT nValue

Wert

STRING* pHex

Zeiger auf den String, der gefüllt werden soll

UDINT nHexSize

Größe des Strings, der gefüllt werden soll

BOOL bWithPraefix

1= Präfix „0x“ wird vorangestellt

Rückgabe:

UDINT

eBRB_ERR_OK = 0

Beschreibung:

Wandelt einen UDINT-Wert in eine hexadezimale Zeichenfolge mit dem optionalen Präfix „0x“.

4.7.9 BrbHexToUdint

```
unsigned long BrbHexToUdint(plcstring* pHex)
```

Argumente:

STRING* pHex

Zeiger auf den String, der gewandelt werden soll

Rückgabe:

UDINT

Ergebnis

Beschreibung:

Wandelt eine Hex-Zeichenfolge in einen UDINT-Wert. Ein Präfix darf nicht vorhanden sein. Ist der im Text angegebene Wert zu groß für UDINT, wird der maximale Wert (4294967295) zurückgegeben.

4.7.10 BrbUdintToBin

```
unsigned short BrbUdintToBin(unsigned long nValue, plcstring* pBin, unsigned long nBinSize, plcbit bWithPraefix)
```

Argumente:

`UDINT` nValue
Wert
`STRING*` pBin
Zeiger auf den String, der gefüllt werden soll
`UDINT` nBinSize
Größe des Strings, der gefüllt werden soll
`BOOL` bWithPraefix
1= Präfix „%“ wird vorangestellt

Rückgabe:

`UINT`
eBRB_ERR_OK = 0

Beschreibung:

Wandelt einen UDINT-Wert in eine binäre Zeichenfolge mit dem optionalen Präfix „%“.
Jedes False-Bit wird zu „0“, jedes True-Bit zu „1“.

4.7.11 BrbDintToHex

```
unsigned short BrbDintToHex(signed long nValue, plcstring* pHex, unsigned long nHexSize, plcbit bWithPraefix)
```

Argumente:

`DINT` nValue
Wert
`STRING*` pHex
Zeiger auf den String, der gefüllt werden soll
`UDINT` nHexSize
Größe des Strings, der gefüllt werden soll
`BOOL` bWithPraefix
1= Präfix „0x“ wird vorangestellt

Rückgabe:

`UINT`
eBRB_ERR_OK = 0

Beschreibung:

Wandelt einen DINT-Wert in eine hexadezimale Zeichenfolge mit dem optionalen Präfix „0x“.
Wertigkeit:

Dez	Hex
-2147483648	0x80000000
-1	0xFFFFFFFF
0	0x00000000
+1	0x00000001
+2147483647	0x7FFFFFFF

4.7.12 BrbHexToDint

```
signed long BrbHexToDint(plcstring* pHex)
```

Argumente:

`STRING*` pHex
Zeiger auf den String, der gewandelt werden soll

Rückgabe:

`DINT`
Ergebnis

Beschreibung:

Wandelt eine Hex-Zeichenfolge in einen DINT-Wert. Ein Präfix darf nicht vorhanden sein.
Die Interpretation eines Hex-Werts als Wert mit Vorzeichen hängt von der maximal darstellbaren Wertigkeit und deshalb von der Anzahl der Hex-Stellen ab:
Beispiel 2 Stellen (mögliche Wertigkeit: -128..+127):

Hex	Dez
0x80	-128

0xFE	-2
0xFF	-1
0x00	0
0x7F	+127
Beispiel 4 Stellen (mögliche Wertigkeit: -32768..+32767):	
Hex	Dez
0x8000	-32768
0xFFFFE	-2
0xFFFF	-1
0x0000	0
0x0080	128
0x7FFF	+32767
Beispiel 8 Stellen (mögliche Wertigkeit: - 2147483648..+2147483647):	
Hex	Dez
0x80000000	-2147483648
0xFFFFFFFFE	-2
0xFFFFFFFF	-1
0x00000000	0
0x00000080	128
0x7FFFFFFF	+2147483647

Wie aus den Beispielen ersichtlich, muss also ,0x0080‘ anders interpretiert werden als ,0x80‘, weil der Übergang vom positiven in den negativen Bereich verschoben ist.
Die Funktion berücksichtigt daher die Anzahl der Hex-Stellen zur Umrechnung.
Ist der im Text angegebene Wert zu groß für DINT, wird -1 zurückgegeben.

4.7.13 BrbAsciiFieldToString

```
plcbit BrbAsciiFieldToString(unsigned char* pAsciiField, unsigned long nAsciiFieldLen, unsigned long nFinalAsciiCharCount, plcstring* pText, unsigned long nTextSize)
```

Argumente:

USINT* pAsciiField
Zeiger auf das Ascii-Feld
UDINT nAsciiFieldLen
Länge des Ascii-Felds
UDINT nFinalAsciiCharCount
Anzahl der letzten Ascii-Zeichen, die auf jeden Fall enthalten sein sollten
STRING* pText
Zeiger auf den String, der den Text aufnehmen soll
UDINT nTextSize
Größe der String-Variablen

Rückgabe:

BOOL

0=Alle Zeichen passten in den Text; 1=Nicht alle Zeichen passten in den Text, es musste gekürzt werden

Beschreibung:

Wandelt ein Ascii-Feld in einen lesbaren Text um. Diese Funktion wird z.B. benutzt, um ein Ascii-Feld mit Steuerzeichen, welches über eine Kommunikation empfangen wird, zum Debuggen als lesbaren Text anzuzeigen.

Lesbare Ascii-Zeichen (Buchstaben und Zahlen, Ascii-Wert >= 32) werden übernommen.

Steuerzeichen (Ascii-Wert <= 31) werden in eine Zeichenfolge gewandelt:

10 wird zu „<10=LF>“

13 wird zu „<13=CR>“

Steuerzeichen, die sich wiederholen werden als Multiplikation angezeigt, um Platz zu sparen:

4 mal 9 wird zu „<4*9=HT>“

Reicht der String nicht aus, um alle Zeichen zu wandeln, werden die letzten Zeichen nicht gewandelt, sondern lediglich die Anzahl angegeben (Kürzung), z.B.:

<+24> wenn die letzten 24 Zeichen keinen Platz mehr hatten.

Mit der Angabe „nFinalAsciiCharCount“ kann die Anzahl der letzten Zeichen angegeben werden, welche auf jeden Fall enthalten sein sollten. Die Anzahl der nicht gewandelten Zeichen (Kürzung „<+x>“) befindet sich dann nicht am Ende, sondern in der Mitte des Strings gefolgt von den letzten gewandelten Zeichen. Damit können z.B. bei Telegrammen die Ende-Zeichen auf jeden Fall angezeigt werden, egal wie lange das Ascii-Feld ist. Es sollte auf jeden Fall darauf geachtet werden, dass der String lang genug ist, um dem Ascii-Feld entsprechend Umwandlungen aufzunehmen.

4.7.14 BrbStringGetIndexOf + BrbWcStringGetIndexOf

```
signed long BrbStringGetIndexOf(plcstring* pText, plcstring* pFind, unsigned long nTextLen)
```

Argumente:

STRING* pText
Zeiger auf den String, der durchsucht werden soll

STRING* pFind
Zeiger auf den String, der gesucht werden soll

UDINT nTextLen
Länge des zu durchsuchenden Textes. Wenn 0, dann wird die Länge ermittelt

Rückgabe:

DINT
Index, an der der Suchtext steht. Wenn nicht gefunden, dann -1

Beschreibung:

Sucht eine Zeichenfolge innerhalb eines Textes und gibt den nullbasierten Index des ersten Vorkommens zurück.

4.7.15 BrbStringGetLastIndexOf + BrbWcStringGetLastIndexOf

```
signed long BrbStringGetLastIndexOf(plcstring* pText, plcstring* pFind, unsigned long nTextLen)
```

Argumente:

STRING* pText
Zeiger auf den String, der durchsucht werden soll

STRING* pFind
Zeiger auf den String, der gesucht werden soll

UDINT nTextLen
Länge des zu durchsuchenden Textes. Wenn 0, dann wird die Länge ermittelt

Rückgabe:

DINT
Index, an der der Suchtext steht. Wenn nicht gefunden, dann -1

Beschreibung:

Sucht eine Zeichenfolge innerhalb eines Textes und gibt den nullbasierten Index des letzten Vorkommens zurück.

4.7.16 BrbStringGetAdrOf + BrbWcStringGetAdrOf

```
plcstring* BrbStringGetAdrOf(plcstring* pText, plcstring* pFind, unsigned long nTextLen)
```

Argumente:

STRING* pText
Zeiger auf den String, der durchsucht werden soll

STRING* pFind
Zeiger auf den String, der gesucht werden soll

UDINT nTextLen
Länge des zu durchsuchenden Textes. Wenn 0, dann wird die Länge ermittelt

Rückgabe:

STRING*
Adresse, an der der Suchtext steht. Wenn nicht gefunden, dann 0

Beschreibung:

Sucht eine Zeichenfolge innerhalb eines Textes und gibt die Adresse des ersten Vorkommens zurück.

4.7.17 BrbStringGetLastAdrOf + BrbWcStringGetLastAdrOf

```
plcstring* BrbStringGetLastAdrOf(plcstring* pText, plcstring* pFind, unsigned long nTextLen)
```

Argumente:

`STRING*` pText
Zeiger auf den String, der durchsucht werden soll

`STRING*` pFind
Zeiger auf den String, der gesucht werden soll

`UDINT` nTextLen
Länge des zu durchsuchenden Textes. Wenn 0, dann wird die Länge ermittelt

Rückgabe:

`STRING*`
Adresse, an der der Suchtext steht. Wenn nicht gefunden, dann 0

Beschreibung:

Sucht eine Zeichenfolge innerhalb eines Textes und gibt die Adresse des letzten Vorkommens zurück.

4.7.18 BrbStringStartsWith + BrbWcStringStartsWith

```
plcbit BrbStringStartsWith(plcstring* pText, plcstring* pFind)
```

Argumente:

`STRING*` pText
Zeiger auf den String, der durchsucht werden soll

`STRING*` pFind
Zeiger auf den String, der gesucht werden soll

Rückgabe:

`BOOL`
Ergebnis

Beschreibung:

Ermittelt, ob ein Text mit einer bestimmten Zeichenfolge beginnt.

4.7.19 BrbStringEndsWith + BrbWcStringEndsWith

```
plcbit BrbStringEndsWith(plcstring* pText, plcstring* pFind)
```

Argumente:

`STRING*` pText
Zeiger auf den String, der durchsucht werden soll

`STRING*` pFind
Zeiger auf den String, der gesucht werden soll

Rückgabe:

`BOOL`
Ergebnis

Beschreibung:

Ermittelt, ob ein Text mit einer bestimmten Zeichenfolge endet.

4.7.20 BrbStringGetSubText + BrbWcStringGetSubText

```
plcstring* BrbStringGetSubText(plcstring* pText, unsigned long nIndex, unsigned long nLen, plcstring* pSubText)
```

Argumente:

`STRING*` pText
Zeiger auf den Anfang des Textes

`UDINT` nIndex
Start-Index des Sub-Textes

`UDINT` nLen
Länge des Sub-Textes

`STRING*` pSubText
Zeiger auf den String, der den Sub-Text aufnimmt

Rückgabe:

`STRING*`
Adresse nach dem Sub-Text

Beschreibung:
Gibt eine Zeichenfolge innerhalb eines Textes zurück.

4.7.21 BrbStringGetSubTextByLen + BrbWcStringGetSubTextByLen

```
plcstring* BrbStringGetSubTextByLen(plcstring* pStart, unsigned long nLen, plcstring* pSubText)
```

Argumente:

- `STRING*` `pStart`
Zeiger auf den Anfang des Sub-Textes
- `unsigned long` `nLen`
Länge des Sub-Textes
- `STRING*` `pSubText`
Zeiger auf den String, der den Sub-Text aufnimmt

Rückgabe:
`STRING*`
Adresse nach dem Sub-Text

Beschreibung:
Gibt eine Zeichenfolge innerhalb eines Textes zurück.

4.7.22 BrbStringGetSubTextByAdr + BrbWcStringGetSubTextByAdr

```
plcstring* BrbStringGetSubTextByAdr(plcstring* pStart, plcstring* pEnd, plcstring* pSubText)
```

Argumente:

- `STRING*` `pStart`
Zeiger auf den Anfang des Sub-Textes
- `STRING*` `pEnd`
Zeiger auf das Ende des Sub-Textes
- `STRING*` `pSubText`
Zeiger auf den String, der den Sub-Text aufnimmt

Rückgabe:
`STRING*`
Adresse nach dem Sub-Text. Wenn ungültige Angaben, dann 0

Beschreibung:
Gibt eine Zeichenfolge innerhalb eines Textes zurück.

4.7.23 BrbStringAppend + BrbWcStringAppend

```
unsigned long BrbStringAppend(plcstring* pText, plcstring* pTextAppend, unsigned long* pTextOffset)
```

Argumente:

- `STRING*` `pText`
Zeiger auf den Anfang des Textes
- `STRING*` `pTextAppend`
Zeiger auf den Anfang des anzuhängenden Textes
- `UDINT*` `pTextOffset`
Zeiger auf den Offset, ab dem angehängt wird

Rückgabe:
`UDINT`
Länge der angehängter Zeichenfolge

Beschreibung:
Hängt eine Zeichenfolge an einen Text und ersetzt somit den klassischen „strcat“ bzw „brwscat“. Bei den Original-Funktionen wird jedesmal die Länge des Basis-Textes ermittelt, um anhängen zu können. Bei sehr großen Texten dauert das entsprechend lange. Wenn jetzt in einer Routine sehr viele strcats hintereinander ausgeführt werden, exponiert die Dauer.
Hier wird das umgangen, indem über „TextOffset“ das Ende des Basis-Textes übergeben wird. Dieser Offset wird in der Funktion erhöht, so dass gleich der nächste Aufruf erfolgen kann.

Nach dem Anhängen wird automatisch eine Null-Terminierung gesetzt.
Vor dem ersten Aufruf sollte ein komplettes Ablöschen des ganzen Basis-Textes auf 0 erfolgen (memset). Enthält der Basis-Text schon Zeichen, muss „TextOffset“ auf die Länge dieses Textes gesetzt werden.

4.7.24 BrbStringCut + BrbWcStringCut

```
plcstring* BrbStringCut(plcstring* pText, unsigned long nCutIndex, unsigned long nCutLen, plcstring* pCut)
```

Argumente:

STRING* pText
Zeiger auf den Anfang des Textes
UDINT nCutIndex
Start-Index der herauszuschneidenden Zeichenfolge
UDINT nCutLen
Länge der herauszuschneidenden Zeichenfolge
STRING* pCut
Zeiger auf den String, der die herausgeschnittene Zeichenfolge aufnimmt. Wenn nicht nötig, dann 0

Rückgabe:

STRING*
Adresse nach der herausgeschnittenen Zeichenfolge

Beschreibung:

Schneidet eine Zeichenfolge aus einem Text heraus.

4.7.25 BrbStringCutFromLastSeparator

```
unsigned long BrbStringCutFromLastSeparator(plcstring* pText, plcstring* pSeparator, plcstring* pCut)
```

Argumente:

STRING* pText
Zeiger auf den Anfang des Textes
STRING* pSeparator
Zeiger auf das zu suchende Trennzeichen (kann auch eine Zeichenfolge sein)
STRING* pCut
Zeiger auf den String, der die abgeschnittene Zeichenfolge aufnimmt. Wenn nicht nötig, dann 0

Rückgabe:

UDINT
Länge der abgeschnittenen Zeichenfolge

Beschreibung:

Sucht von hinten ein Trennzeichen und schneidet ab diesem den Text ab.
Es kann dazu verwendet werden, bei einem Datei-Pfad mit dem Trennzeichen „/“ die letzte Pfadangabe zu löschen und so das übergeordnete Verzeichnis zu bekommen.

4.7.26 BrbStringInsert + BrbWcStringInsert

```
plcstring* BrbStringInsert(plcstring* pText, unsigned long nInsertIndex, plcstring* pInsert)
```

Argumente:

STRING* pText
Zeiger auf den Anfang des Textes
UDINT nInsertIndex
Index, bei dem eingefügt wird
STRING* pInsert
Zeiger auf den String, der die einzufügende Zeichenfolge enthält

Rückgabe:

STRING*
Adresse nach der eingefügten Zeichenfolge

Beschreibung:

Fügt eine Zeichenfolge in einen Text ein.

4.7.27 BrbStringReplace + BrbWcStringReplace

```
unsigned long BrbStringReplace(plcstring* pText, plcstring* pFind, plcstring* pReplace)
```

Argumente:

`STRING*` pText
Zeiger auf den Anfang des Textes
`STRING*` pFind
Zeiger auf die zu ersetzenden Zeichenfolge
`STRING*` pReplace
Zeiger auf die ersetzende Zeichenfolge

Rückgabe:

`UDINT`
Anzahl, wie oft ersetzt wurde

Beschreibung:

Ersetzt eine Zeichenfolge in einem Text durch eine andere.

4.7.28 BrbStringPadLeft + BrbWcStringPadLeft

```
plcbit BrbStringPadLeft(plcstring* pText, plcstring* pFillChar, unsigned long nLen)
```

Argumente:

`STRING*` pText
Zeiger auf den String, der gefüllt werden soll
`STRING*` pFillChar
Zeiger auf den String, mit dem gefüllt werden soll
`UDINT` nLen
Länge, auf die gefüllt werden soll

Rückgabe:

`BOOL`
0=String konnte nicht gefüllt werden
1=String konnte gefüllt werden

Beschreibung:

Füllt einen Text auf der linken Seite mit einem Füllzeichen auf eine bestimmte Gesamtlänge auf.

4.7.29 BrbStringPadRight + BrbWcStringPadRight

```
plcbit BrbStringPadRight(plcstring* pText, plcstring* pFillChar, unsigned long nLen)
```

Argumente:

`STRING*` pText
Zeiger auf den String, der gefüllt werden soll
`STRING*` pFillChar
Zeiger auf den String, mit dem gefüllt werden soll
`UDINT` nLen
Länge, auf die gefüllt werden soll

Rückgabe:

`BOOL`
0=String konnte nicht gefüllt werden
1=String konnte gefüllt werden

Beschreibung:

Füllt einen Text auf der rechten Seite mit einem Füllzeichen auf eine bestimmte Gesamtlänge auf.

4.7.30 BrbStringTrimLeft + BrbWcStringTrimLeft

```
plcbit BrbStringTrimLeft(plcstring* pText, plcstring* pTrim)
```

Argumente:

`STRING*` pText
Zeiger auf den String, der getrimmt werden soll
`STRING*` pTrim
Zeiger auf den String, mit dem getrimmt werden soll

Rückgabe:

BOOL

0=String wurde nicht getrimmt
1=String wurde getrimmt

Beschreibung:

Schneidet eine wiederkehrende Zeichenfolge auf der linken Seite eines Textes heraus.
Damit können z.B. mehrere Leerzeichen am Beginn eines Strings entfernt werden.

4.7.31 BrbStringTrimRight + BrbWcStringTrimRight

```
plcbit BrbStringTrimRight(plcstring* pText, plcstring* pTrim)
```

Argumente:

STRING* pText
Zeiger auf den String, der getrimmt werden soll
STRING* pTrim
Zeiger auf den String, mit dem getrimmt werden soll

Rückgabe:

BOOL
0=String wurde nicht getrimmt
1=String wurde getrimmt

Beschreibung:

Schneidet eine wiederkehrende Zeichenfolge auf der rechten Seite eines Textes heraus.
Damit können z.B. mehrere Leerzeichen am Ende eines Strings entfernt werden.

4.7.32 BrbStringSplit

```
unsigned long BrbStringSplit(plcstring* pText, plcstring* pSep, unsigned short pSplitArray, unsigned long nArrayIndexMax, unsigned long nEntrySize)
```

Argumente:

STRING* pText
Zeiger auf den String, der gesplittet werden soll
STRING* pSep
Zeiger auf den String, der das Trennzeichen enthält
UDINT pSplitArray
Zeiger auf das String-Array, das die gesplitteten Texte aufnimmt
UDINT nArrayIndexMax
Maximaler Index des String-Arrays
UDINT nEntrySize
Größe eines Strings im String-Array

Rückgabe:

UDINT
Anzahl der einzelnen Strings

Beschreibung:

Spaltet einen Text aufgrund eines Trennzeichens in mehrere Strings.
Damit kann eine Liste in einem String in einzelne Texte gespalten werden. Das Trennzeichen wird nicht in das String-Array übernommen.
Leere Strings werden nicht in die Liste aufgenommen und auch nicht gezählt.
Es ist darauf zu achten, dass das Array und auch dessen Elemente groß genug sind, um das Ergebnis aufnehmen zu können.

4.7.33 BrbStringSplitEmpty

```
unsigned long BrbStringSplitEmpty (plcstring* pText, plcstring* pSep, unsigned long pSplitArray, unsigned long nArrayIndexMax, unsigned long nEntrySize)
```

Argumente:

STRING* pText
Zeiger auf den String, der gesplittet werden soll
STRING* pSep
Zeiger auf den String, der das Trennzeichen enthält
UDINT pSplitArray
Zeiger auf das String-Array, das die gesplitteten Texte aufnimmt

`UDINT` `nArrayIndexMax`
Maximaler Index des String-Arrays
`UDINT` `nEntrySize`
Größe eines Strings im String-Array

Rückgabe:

`UDINT`
Anzahl der einzelnen Strings

Beschreibung:

Verhält sich wie ‚BrbStringSplit‘. Im Gegensatz dazu werden aber leere Strings in die Liste aufgenommen und mitgezählt.

Es ist darauf zu achten, dass das Array und auch dessen Elemente groß genug sind, um das Ergebnis aufnehmen zu können.

4.7.34 BrbStringConvertRealFromExp

```
plcbit BrbStringConvertRealFromExp(plcstring* pValue, plcstring* pResult, unsigned long nResultSize)
```

Argumente:

`STRING*` `pValue`
Zeiger auf den String, der gewandelt werden soll
`STRING*` `pResult`
Zeiger auf den String, der das Ergebnis aufnimmt
`UDINT` `nResultSize`
Größe des Ergebnis-Strings

Rückgabe:

`UINT`
`eBRB_ERR_OK = 0`

Beschreibung:

Wandelt eine exponentielle Notation („1.23e-3“) in einen Real-Wert („0.00123“).

Ist es schon ein Real-Wert, werden führende und folgende Nullen herausgeschnitten.

4.7.35 BrbStringConvertRealToExp

```
plcbit BrbStringConvertRealToExp(plcstring* pValue, plcstring* pResult, unsigned long nResultSize)
```

Argumente:

`STRING*` `pValue`
Zeiger auf den String, der gewandelt werden soll
`STRING*` `pValue`
Zeiger auf den String, der das Ergebnis aufnimmt
`UDINT` `nResultSize`
Größe des Ergebnis-Strings

Rückgabe:

`UINT`
`eBRB_ERR_OK = 0`

Beschreibung:

Wandelt einen Real-Wert („0.00123“) in eine exponentielle Notation („1.23e-003“).

Ist es schon eine Notation, werden folgende Nullen herausgeschnitten und der Exponent auf mindestens 3 Stellen erweitert.

4.7.36 BrbStringFormatFractionDigits

```
plcbit BrbStringFormatFractionDigits(plcstring* pValue, unsigned long nValueSize, unsigned short nFractionsDigits)
```

Argumente:

`STRING*` `pValue`
Zeiger auf den String, der formatiert werden soll
`UDINT` `nValueSize`
Größe des Strings

`UINT nFractionsDigits`
Anzahl der gewünschten Nachkommastellen

Rückgabe:

`UINT`
`eBRB_ERR_OK = 0`

Beschreibung:

Fügt Nachkommastellen bei einem Text, der eine Real-Zahl enthält, an oder schneidet sie ab.

4.7.37 BrbStringSwap

`plcbit BrbStringSwap(plcstring* pText, plcstring* pSwapped, unsigned long nSwappedSize)`

Argumente:

`STRING*` `pText`
Zeiger auf den String, der umgedreht werden soll
`STRING*` `pSwapped`
Zeiger auf den String, der den umgedrehten Text aufnimmt
`UDINT` `nSwappedSize`
Größe des Strings, der den umgedrehten Text aufnimmt

Rückgabe:

`BOOL`
0=Text konnte nicht gedreht werden
1=Text konnte gedreht werden

Beschreibung:

Dreht die Reihenfolge der Zeichen in einem Text.

4.7.38 BrbStringToUpper + BrbWcStringToUpper

`plcbit BrbStringToUpper(plcstring* pText)`

Argumente:

`STRING*` `pText`
Zeiger auf den String

Rückgabe:

`BOOL`
0=Text enthielt keine Kleinbuchstaben
1=Text enthielt Kleinbuchstaben

Beschreibung:

Wandelt alle Buchstaben zu Großbuchstaben.

4.7.39 BrbStringToLower + BrbWcStringToLower

`plcbit BrbStringToLower(plcstring* pText)`

Argumente:

`STRING*` `pText`
Zeiger auf den String

Rückgabe:

`BOOL`
0=Text enthielt keine Großbuchstaben
1=Text enthielt Großbuchstaben

Beschreibung:

Wandelt alle Buchstaben zu Kleinbuchstaben.

4.7.40 BrbStringIsNumerical + BrbWcStringIsNumerical

`plcbit BrbStringIsNumerical(plcstring* pText)`

Argumente:

`STRING*` `pText`
Zeiger auf den String

Rückgabe:

BOOL

0=Text enthält keinen numerischen Wert
1= Text enthält numerischen Wert

Beschreibung:

Gibt zurück, ob ein Text einen numerischen Wert enthält. Dabei werden auch Zahlen als exponentielle Notation erkannt. Folgende Zeichen dürfen enthalten sein:

+ Nur am Anfang oder nach „e“
- Nur am Anfang oder nach „e“
e Nur einmal
, Nur einmal, aber nicht zusammen mit .
. Nur einmal, aber nicht zusammen mit ,
0-9

4.7.41 BrbStringIsHex + BrbWcStringIsHex

```
plcbit BrbStringIsHex(plcstring* pText, plcbit bLowerCaseAllowed)
```

Argumente:

STRING* pText

Zeiger auf den String

BOOL bLowerCaseAllowed

Gibt an, ob auch die Kleinbuchstaben „a..f“ erlaubt sind

Rückgabe:

BOOL

0=Text enthält keinen hexadezimalen Wert
1= Text enthält hexadezimalen Wert

Beschreibung:

Gibt zurück, ob ein Text einen hexadezimalen Wert enthält.
Folgende Zeichen dürfen enthalten sein:

0-9
A-F
a-f Nur wenn „bLowerCaseAllowed“ = 1

4.7.42 BrbStringCountText + BrbWcStringCountText

```
unsigned long BrbStringCountText(plcstring* pText, plcstring* pFind, unsigned long nTextLen)
```

Argumente:

STRING* pText

Zeiger auf den String, der durchsucht werden soll

STRING* pFind

Zeiger auf den String, der gesucht werden soll

UDINT nTextLen

Länge des zu durchsuchenden Textes. Wenn 0, dann wird die Länge ermittelt

Rückgabe:

UDINT

Anzahl der gefundenen Zeichenfolge

Beschreibung:

Gibt die Anzahl des Vorkommens einer Zeichenfolge innerhalb eines Textes zurück.

4.7.43 BrbStringRepeat + BrtWcStringRepeat

```
unsigned long BrbStringRepeat(plcstring* pText, unsigned long nTextSize, plcstring* pRepeat, unsigned long nLen)
```

Argumente:

STRING* pText

Zeiger auf den aufzufüllenden Text

UDINT nTextSize

Gesamt-Länge des aufzufüllenden Textes
`STRING*` `pRepeat` Zeiger auf die zu wiederholende Zeichenfolge
`UDINT` `nLen` Anzahl der Zeichen, auf die wiederholend aufgefüllt wird

Rückgabe:

`UDINT`
Anzahl der angehängten Zeichen

Beschreibung:

Diese Funktion hängt eine Zeichenfolge solange wiederholend an einen Text, bis eine vorgegebene Gesamt-Anzahl an Zeichen erreicht ist. Zum Schutz muss die Größe des Textes angegeben werden.
Beispiel:

```
pText = „Michael“
nTextSize = sizeof(pText)
pRepeat = „Abcd“
nLen = 14

      Text-Länge   12345678901234
Ergebnis:      MichaelAbcdAbc
```

4.7.44 BrbStringCopy + BrbWcStringCopy

```
unsigned long BrbStringCopy(plcstring* pDest, plcstring* pSrc, unsigned long nDestSize)
```

Argumente:

`STRING*` `pDest` Zeiger auf den Zielstring
`STRING*` `pSrc` Zeiger auf den Quellstring
`UDINT` `nDestSize` Größe des Zielstrings

Rückgabe:

`UDINT`
Anzahl der zu kopierenden Zeichen

Beschreibung:

Diese Funktion garantiert ein sicheres Kopieren eines Strings und umgeht damit die Risiken der herkömmlichen Ansi-C-Befehle.

Beschreibung der herkömmlichen Ansi-C-Befehle:

```
strcpy(pDest, pSrc)
```

Kopiert Zeichen bis zur Erkennung des Null-Terminators im Quellstring

Risiko: Ein Quelltext wird ohne Prüfung auf einen zu kurzen Zielstring kopiert und schreibt deshalb über dessen Speicherbereich hinaus!

```
strncpy(pDest, pSrc, nSize)
```

Kopiert Zeichen maximal bis zur übergebenen Größe, berücksichtigt also den Speicherbereich des Zielstrings. Anschließend wird aber kein expliziter Null-Terminator gesetzt.

Risiko: Ein zu langer Quellstring wird zwar abgeschnitten kopiert, aber es wird nicht sichergestellt, dass der Zielstring mit einem Null-Terminator abgeschlossen ist!

```
strncpy(pDest, pSrc, nSize)
```

Kopiert Zeichen maximal bis zur übergebenen Größe und setzt den Null-Terminator am Zielstring, füllt den Zielstring aber nicht mit 0 auf.

Diesen Befehl gibt es nur auf einigen Plattformen (z.B. OpenBSD), nicht jedoch im bei B&R verwendeten GNU-Compiler, er steht also bei B&R nicht zur Verfügung!

Diese Funktion dagegen bietet größtmögliche Sicherheit. Sie geht wie folgt vor:

Der Quellstring wird unter Berücksichtigung der Speichergröße des Zielstrings kopiert. Das heißt, es werden maximal so viel Zeichen kopiert, wie im Zielstring Platz haben. Der evtl. restliche Speicherbereich wird mit 0 aufgefüllt. Der Null-Terminator wird aber auf jeden Fall gesetzt.

Hinweis: Durch das Auffüllen des Zielstrings mit 0 dauert der Aufruf dieser Funktion abhängig von der Zielstring-Größe länger als die herkömmlichen Befehle, merzt aber das Risiko eines Speicher-Schmierers und „unsauberer“ Strings aus.

Ist der Vorgang nicht möglich (wegen Null-Pointer usw.), wird 0 zurückgegeben.

4.7.45 BrbStringCat + BrbWcStringCat

```
unsigned long BrbStringCat (plcstring* pDest, plcstring* pSrc, unsigned long nDestSize)
```

Argumente:

`STRING*` pDest
Zeiger auf den Zielstring
`STRING*` pSrc
Zeiger auf den Quellstring
`UDINT` nDestSize
Größe des Zielstrings

Rückgabe:

`UDINT`
Anzahl der zu erreichenden Zeichen bei vollständiger Verkettung

Beschreibung:

Diese Funktion garantiert ein sicheres Verketteten von Strings und umgeht damit die Risiken der herkömmlichen Ansi-C-Befehle.

Beschreibung der herkömmlichen Ansi-C-Befehle:

`strcat (pDest, pSrc)`

Verkettet die Zeichen bis zur Erkennung des Null-Terminators im Quellstring
Risiko: Ein Quelltext wird ohne Prüfung auf einen zu kurzen Zielstring angehängt und schreibt deshalb über dessen Speicherbereich hinaus!

`strncat (pDest, pSrc, nSize)`

Verkettet die Zeichen maximal bis zur übergebenen Größe, berücksichtigt also den Speicherbereich des Zielstrings. Anschließend wird aber kein expliziter Null-Terminator gesetzt.

Risiko: Ein zu langer Quellstring wird zwar abgeschnitten angehängt, aber es wird nicht sichergestellt, dass der Zielstring mit einem Null-Terminator abgeschlossen ist!

`strlcat (pDest, pSrc, nSize)`

Verkettet die Zeichen maximal bis zur übergebenen Größe und setzt den Null-Terminator am Zielstring, füllt den Zielstring aber nicht mit 0 auf.

Diesen Befehl gibt es nur auf einigen Plattformen (z.B. OpenBSD), nicht jedoch im bei B&R verwendeten GNU-Compiler, er steht also bei B&R nicht zur Verfügung!

Diese Funktion dagegen bietet größtmögliche Sicherheit. Sie geht wie folgt vor:

Der Quellstring wird unter Berücksichtigung der Speichergröße an den Zielstrings angehängt. Das heißt, es werden maximal so viel Zeichen angehängt, wie im Zielstring Platz haben. Der evtl. restliche Speicherbereich wird mit 0 aufgefüllt. Der Null-Terminator wird aber auf jeden Fall gesetzt.

Hinweis: Durch das Auffüllen des Zielstrings mit 0 dauert der Aufruf dieser Funktion abhängig von der Zielstring-Größe länger als die herkömmlichen Befehle, merzt aber das Risiko eines Speicher-Schmierers und „unsauberer“ Strings aus.

Ist der Vorgang nicht möglich (wegen Null-Pointer usw.), wird 0 zurückgegeben.

4.8 BigDatatypes

Die größten auf der Sps unter IEC unterstützten Integer-Datentypen sind UDINT (Uint32) und DINT (Int32).

Manchmal werden aber auch größere Datentypen als Zähler benötigt (z.B. bei OpcUa).

Dieses Paket unterstützt bei der Implementierung von Uint64 und Int64.

Achtung: Es sind unbedingt die [Hinweise zur Konsistenz](#) am Ende des Kapitels zu beachten!

4.8.1 Uint64 (ULINT)

Hier kann ein vorzeichenloser 64-Bit-Wert implementiert werden. Der Wertebereich umfasst 0...18446744073709551615.

4.8.1.1 Struktur

Zur Ablage eines Uint64-Wertes wird diese Struktur verwendet:

BrbUint64_TYP		
nLow	UDINT	Niederwertige Bits
nHigh	UDINT	Höherwertige Bits

Sie enthält die notwendigen 64 Bits.

4.8.1.2 BrbUint64FromString

```
signed long BrbUint64FromString(struct BrbUint64_TYP* pUint64, plcstring* pText)
```

Argumente:

`struct BrbUint64_TYP* pUint64`
Zeiger auf die Uint64-Struktur
`STRING* pText`
Zeiger auf den String

Rückgabe:

`DINT`
`eBRB_ERR_OK = 0`
`eBRB_ERR_NULL_POINTER = 50000`
`eBRB_ERR_OVERFLOW = 50003`
`eBRB_ERR_STRING_TOO_LONG = 50006`

Beschreibung:

Wandelt eine Zeichenfolge in eine Uint64-Struktur. So können große Werte bequem gesetzt werden. Ein zu langer Text oder ein Überlauf wird erkannt und als Status zurückgeben.

Zur Deklaration des Strings steht die Konstante `nBRB_UINT64_CHAR_MAX = 21` zur Verfügung.

4.8.1.3 BrbUint64ToString

```
signed long BrbUint64ToString(struct BrbUint64_TYP* pUint64, plcstring* pText, unsigned long nTextSize)
```

Argumente:

`struct BrbUint64_TYP* pUint64`
Zeiger auf die Uint64-Struktur
`STRING* pText`
Zeiger auf den String, der gefüllt werden soll
`UDINT nTextSize`
Größe des Strings

Rückgabe:

`DINT`
`eBRB_ERR_OK = 0`
`eBRB_ERR_NULL_POINTER = 50000`
`eBRB_ERR_STRING_TOO_SHORT = 50005`

Beschreibung:

Wandelt eine Uint64-Struktur in eine Zeichenfolge. So kann der Wert bequem als String angezeigt werden.

Achtung: Es ist auf ausreichende Größe der Strings zu achten. Zur Deklaration des Strings steht die Konstante `nBRB_UINT64_CHAR_MAX = 21` zur Verfügung.

4.8.1.4 BrbUint64Compare

```
signed char BrbUint64Compare(struct BrbUint64_TYP* pOperand0, struct BrbUint64_TYP* pOperand1)
```

Argumente:

```
struct BrbUint64_TYP* pOperand0  
    Zeiger auf die erste Uint64-Struktur  
struct BrbUint64_TYP* pOperand1  
    Zeiger auf die zweite Uint64-Struktur
```

Rückgabe:

```
SINT  
-1 = Erster Wert ist kleiner als der zweite  
0 = Erster Wert ist gleich dem zweiten  
1 = Erster Wert ist größer als der zweite
```

Beschreibung:

Vergleicht zwei Uint64-Strukturen auf kleiner, größer oder gleich.

4.8.1.5 BrbUint64Inc

```
signed long BrbUint64Inc(struct BrbUint64_TYP* pUint64)
```

Argumente:

```
struct BrbUint64_TYP* pUint64  
    Zeiger auf die Uint64-Struktur
```

Rückgabe:

```
DINT  
eBRB_ERR_OK = 0  
eBRB_ERR_NULL_POINTER = 50000
```

Beschreibung:

Inkrementiert einen Uint64-Wert um 1. Ein Überlauf tritt mit analogen Regeln auf wie bei UDINT.

4.8.1.6 BrbUint64Dec

```
signed long BrbUint64Dec(struct BrbUint64_TYP* pUint64)
```

Argumente:

```
struct BrbUint64_TYP* pUint64  
    Zeiger auf die Uint64-Struktur
```

Rückgabe:

```
DINT  
eBRB_ERR_OK = 0  
eBRB_ERR_NULL_POINTER = 50000
```

Beschreibung:

Dekrementiert einen Uint64-Wert um 1. Ein Unterlauf tritt mit analogen Regeln auf wie bei UDINT.

4.8.1.7 BrbUint64Calculate

```
signed long BrbUint64Calculate(struct BrbUint64_TYP* pOperand0, struct BrbUint64_TYP* pOperand1,  
enum BrbDt64Operator_ENUM eOperator)
```

Argumente:

```
struct BrbUint64_TYP* pOperand0  
    Zeiger auf die erste Uint64-Struktur  
struct BrbUint64_TYP* pOperand1  
    Zeiger auf die zweite Uint64-Struktur  
BrbDt64Operator_ENUM eOperator  
    Operator der Rechnung
```

BrbDt64Operator_ENUM		
eBRB_DT64_OPERATOR_ADD		Addieren
eBRB_DT64_OPERATOR_SUBTRACT		Subtrahieren
eBRB_DT64_OPERATOR_MULTIPLY		Multiplizieren
eBRB_DT64_OPERATOR_DIVIDE		Dividieren

Rückgabe:

DINT

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

Beschreibung:

Führt eine Berechnung mit zwei Uint64 aus. Das Ergebnis wird im ersten Uint64 abgelegt. Ein Über- oder Unterlauf tritt mit analogen Regeln auf wie bei UDINT.

4.8.2 Int64 (LINT)

Hier kann ein vorzeichenbehafteter 64-Bit-Wert implementiert werden. Der Wertebereich umfasst -9223372036854775808...+9223372036854775807.

4.8.2.1 Struktur

Zur Ablage eines Int64-Wertes wird diese Struktur verwendet:

BrbInt64_TYP		
nLow	UDINT	Niederwertige Bits
nHigh	UDINT	Höherwertige Bits

Sie enthält die notwendigen 64 Bits.

4.8.2.2 BrbInt64FromString

```
signed long BrbInt64FromString(struct BrbInt64_TYP* pInt64, plcstring* pText)
```

Argumente:

```
struct BrbInt64_TYP* pInt64  
    Zeiger auf die Int64-Struktur  
STRING* pText  
    Zeiger auf den String
```

Rückgabe:

DINT

```
eBRB_ERR_OK = 0  
eBRB_ERR_NULL_POINTER = 50000  
eBRB_ERR_OVERFLOW = 50003  
eBRB_ERR_STRING_TOO_LONG = 50006
```

Beschreibung:

Wandelt eine Zeichenfolge in eine Int64-Struktur. So können große Werte bequem gesetzt werden. Negative Werte beginnen mit einem „-“. Ein zu langer Text oder ein Überlauf wird erkannt und als Status zurückgeben. Zur Deklaration des Strings steht die Konstante nBRB_INT64_CHAR_MAX = 21 zur Verfügung.

4.8.2.3 BrbInt64ToString

```
signed long BrbInt64ToString(struct BrbInt64_TYP* pInt64, plcstring* pText, unsigned long nTextSize)
```

Argumente:

```
struct BrbInt64_TYP* pInt64  
    Zeiger auf die Int64-Struktur  
STRING* pText  
    Zeiger auf den String, der gefüllt werden soll  
UDINT nTextSize  
    Größe des Strings
```

Rückgabe:`DINT`

```
eBRB_ERR_OK = 0
eBRB_ERR_NULL_POINTER = 50000
eBRB_ERR_STRING_TOO_SHORT = 50005
```

Beschreibung:

Wandelt eine Int64-Struktur in eine Zeichenfolge. So kann der Wert bequem als String angezeigt werden.

Achtung: Es ist auf ausreichende Größe der Strings zu achten. Zur Deklaration des Strings steht die Konstante `nBRB_INT64_CHAR_MAX` = 21 zur Verfügung.

4.8.2.4 BrbInt64Compare

```
signed char BrbInt64Compare(struct BrbInt64_TYP* pOperand0, struct BrbInt64_TYP* pOperand1)
```

Argumente:

```
struct BrbInt64_TYP* pOperand0
    Zeiger auf die erste Int64-Struktur
struct BrbInt64_TYP* pOperand1
    Zeiger auf die zweite Int64-Struktur
```

Rückgabe:`SINT`

```
-1 = Erster Wert ist kleiner als der zweite
0 = Erster Wert ist gleich dem zweiten
1 = Erster Wert ist größer als der zweite
```

Beschreibung:

Vergleicht zwei Int64-Strukturen auf kleiner, größer oder gleich.

4.8.2.5 BrbInt64Inc

```
signed long BrbInt64Inc(struct BrbInt64_TYP* pInt64)
```

Argumente:

```
struct BrbInt64_TYP* pInt64
    Zeiger auf die Int64-Struktur
```

Rückgabe:`DINT`

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

Beschreibung:

Inkrementiert einen Int64-Wert um 1. Ein Überlauf tritt mit analogen Regeln auf wie bei DINT.

4.8.2.6 BrbInt64Dec

```
signed long BrbInt64Dec(struct BrbInt64_TYP* pInt64)
```

Argumente:

```
struct BrbInt64_TYP* pInt64
    Zeiger auf die Int64-Struktur
```

Rückgabe:`DINT`

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

Beschreibung:

Dekrementiert einen Int64-Wert um 1. Ein Unterlauf tritt mit analogen Regeln auf wie bei DINT.






4.8.2.7 BrbInt64Calculate

```
signed long BrbInt64Calculate(struct BrbInt64_TYP* pOperand0, struct BrbInt64_TYP* pOperand1,
enum BrbDt64Operator_ENUM eOperator)
```

Argumente:

```
struct BrbInt64_TYP* pOperand0  
    Zeiger auf die erste Int64-Struktur  
struct BrbInt64_TYP* pOperand1  
    Zeiger auf die zweite Int64-Struktur  
BrbDt64Operator_ENUM eOperator
```

Operator der Rechnung

	BrbDt64Operator_ENUM	
	eBRB_DT64_OPERATOR_ADD	Addieren
	eBRB_DT64_OPERATOR_SUBTRACT	Subtrahieren
	eBRB_DT64_OPERATOR_MULTIPLY	Multiplizieren
	eBRB_DT64_OPERATOR_DIVIDE	Dividieren

Rückgabe:

DINT

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

Beschreibung:

Führt eine Berechnung mit zwei Int64 aus. Dabei dürfen der erste und der zweite Operand sowohl positiv als auch negativ sein. Das Ergebnis wird im ersten Int64 abgelegt. Ein Über- oder Unterlauf tritt mit analogen Regeln auf wie bei DINT.

4.8.3 Hilfsfunktionen

4.8.3.1 BrbIntegerStringAdd

```
unsigned short BrbIntegerStringAdd(plcstring* pIntegerString1, plcstring* pIntegerString2, unsigned long nSize1)
```

Argumente:

```
STRING* pIntegerString1  
    Zeiger auf den Anfang des ersten Integer-Strings  
STRING* pIntegerString2  
    Zeiger auf den Anfang des zweiten Integer-Strings  
UDINT nDestSize  
    Größe des ersten Integer-Strings
```

Rückgabe:

DINT

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

Beschreibung:

Addiert zwei Integer-Strings, welche Ganzzahlen enthalten müssen. Das Ergebnis wird im ersten Integer-String abgelegt.

Achtung: Es ist auf ausreichende Größe der Strings zu achten.

Diese Funktion wird als Hilfsfunktion für Uint64 und Int64 verwendet.

4.8.3.2 BrbByteArrayAdd

```
signed long BrbByteArrayAdd(unsigned char* pByteArray0, unsigned long nLength0, unsigned char* pByteArray1, unsigned long nLength1)
```

Argumente:

```
USINT* pByteArray0  
    Zeiger auf den Anfang des ersten Byte-Arrays  
UDINT nLength0  
    Länge des ersten Byte-Arrays in Byte  
USINT* pByteArray1  
    Zeiger auf den Anfang des zweiten Byte-Arrays  
UDINT nLength1  
    Länge des zweiten Byte-Arrays in Byte
```

Rückgabe:

DINT

```
eBRB_ERR_OK = 0
```

```
eBRB_ERR_NULL_POINTER = 50000  
eBRB_ERR_OVERFLOW = 50003
```

Beschreibung:

Addiert zwei Integer-Zahlen, welche als Byte-Array abgelegt sind. Das Ergebnis wird im ersten Byte-Array abgelegt. Ein Überlauf wird erkannt und als Status zurückgeben.

Diese Funktion wird als Hilfsfunktion für Uint64 und Int64 verwendet.

4.8.3.3 BrbByteArraySubtract

```
signed long BrbByteArraySubtract(unsigned char* pByteArray0, unsigned long nLength0, unsigned  
char* pByteArray1, unsigned long nLength1)
```

Argumente:

```
USINT* pByteArray0  
    Zeiger auf den Anfang des ersten Byte-Arrays  
UDINT nLength0  
    Länge des ersten Byte-Arrays in Byte  
USINT* pByteArray1  
    Zeiger auf den Anfang des zweiten Byte-Arrays  
UDINT nLength1  
    Länge des zweiten Byte-Arrays in Byte
```

Rückgabe:

```
DINT  
eBRB_ERR_OK = 0  
eBRB_ERR_NULL_POINTER = 50000  
eBRB_ERR_OVERFLOW = 50003
```

Beschreibung:

Subtrahiert zwei Integer-Zahlen, welche als Byte-Array abgelegt sind. Das Ergebnis wird im ersten Byte-Array abgelegt. Ein Unterlauf wird erkannt.

Diese Funktion wird als Hilfsfunktion für Uint64 und Int64 verwendet.

4.8.4 Hinweise zur Konsistenz

Konsistenz bedeutet, dass alle Bits des Wertes richtig gesetzt sind.

In diesen Funktionen werden zum einen 2*32Bit verarbeitet und zum anderen teilweise auch längerer Code eingesetzt (z.B. Schleifen). Eine Funktion kann somit jederzeit durch eine schnellere Taskklasse unterbrochen werden, obwohl noch nicht alle Bits verarbeitet worden sind. In diesem Fall sind die Bits nicht konsistent zueinander.

Das bedeutet, dass die Konsistenz der 64-Bit-Werte nicht garantiert werden kann! Dies ist vor allem bei globalen Instanzen zu beachten, die in unterschiedlichen Taskklassen verwendet werden!

4.9 Xml

In diesem Paket finden sich Funktionen zum Parsen von Xml-Texten.

4.9.1 BrbXmlGetTagText

```
unsigned long BrbXmlGetTagText(plcstring* pStartTag, plcstring* pEndTag, unsigned long pStart,
unsigned long pEnd, plcstring* pText, unsigned long nTextSize)
```

Argumente:

STRING* pStartTag
Zeiger auf den String, der den Start-Tag enthält

STRING* pEndTag
Zeiger auf den String, der den End-Tag enthält

UDINT pStart
Adresse, an der die Suche nach dem Start-Tag beginnt

UDINT pEnd
Adresse, an der die Suche nach dem Start-Tag endet

STRING* pText
Zeiger auf den String, der den Tag-Wert aufnimmt

UDINT nTextSize
Größe des Strings, der den Tag-Wert aufnimmt

Rückgabe:

UDINT
Adresse nach dem End-Tag. Wenn nicht gefunden, dann 0

Beschreibung:

Sucht nach einem Start- sowie einem End-Tag und gibt den Text dazwischen zurück. Sonder- und Formatierungszeichen (z.B. Zeilenumbruch) werden dabei nicht zurückgegeben.

Beispiel:

```
sXml=
<?xml version="1.0" encoding="utf-8"?>
<Library Name="BrbLib">
  <Functions>
    <Function>
      <Name>BrbUdintToAscii</Name>
      <Arguments>
        <Argument>nValue</Argument>
        <Argument>pText</Argument>
      </Arguments>
    </Function>
    <Function>
      <Name>BrbHexToUdint</Name>
      <Arguments>
        <Argument>pHex</Argument>
      </Arguments>
    </Function>
  </Functions>
</Library>

pStartTag= „<Name>“
pEndTag= „</Name>“
pStart = ADR(sXml)
pEnd = ADR(sXml)+ LEN(sXml)
pText = ADR(sText)
nTextSize = sizeof(sText)
```

Es wird der erste Tag ‚Name‘ gefunden:

Ermittelter Text: sText = "BrbUdintToAscii"

Der Ausgang gibt die unmittelbare Adresse nach dem Tag zurück, zeigt also auf das erste „<Arguments>“.

Jetzt kann der Start auf die zurückgegebene Adresse gesetzt werden. Nach einem erneuten Aufruf wird der zweite Tag ‚Name‘ gefunden

Ermittelter Text: sText = "BrbHexToUdint"

Der Ausgang gibt die unmittelbare Adresse nach dem Tag zurück, zeigt also auf das zweite „<Arguments>“.

Im Wechselspiel mit der Funktion ‚BrbXmlGetNextTag‘ kann so der Xml-Text durchgeparst und alle Daten ermittelt werden.

4.9.2 BrbXmlGetNextTag

```
unsigned long BrbXmlGetNextTag(plcstring* pTag, unsigned long pStart, unsigned long pEnd)
```

Argumente:

`STRING*` pTag
Zeiger auf den String, der den Tag enthält

`UDINT` pStart
Adresse, an der die Suche nach dem Tag beginnt

`UDINT` pEnd
Adresse, an der die Suche nach dem Tag endet

Rückgabe:

`UDINT`
Adresse nach dem Tag. Wenn nicht gefunden, dann 0

Beschreibung:

Sucht nach einem Tag und gibt die Adresse nach diesem Tag zurück.

Beispiel:

```
sXml=
<?xml version="1.0" encoding="utf-8"?>
<Library Name="BrbLib">
  <Functions>
    <Function>
      <Name>BrbUdintToAscii</Name>
      <Arguments>
        <Argument>nValue</Argument>
        <Argument>pText</Argument>
      </Arguments>
    </Function>
    <Function>
      <Name>BrbHexToUdint</Name>
      <Arguments>
        <Argument>pHex</Argument>
      </Arguments>
    </Function>
  </Functions>
</Library>

pTag= „<Arguments>“
pStart = ADR(sXml)
pEnd = ADR(sXml) + LEN(sXml)
```

Der Ausgang gibt die unmittelbare Adresse nach dem Tag zurück, zeigt also auf das erste „<Argument>“.

Jetzt kann der Start auf die zurückgegebene Adresse gesetzt werden.

Der Ausgang gibt die unmittelbare Adresse nach dem Tag zurück, zeigt also auf das zweite „<Argument>“.

Im Wechselspiel mit der Funktion ‚BrbXmlGetTag‘ kann so der Xml-Text durchgeparst und alle Daten ermittelt werden.

4.10 Memory

In diesem Paket finden sich Funktionen zum Verwalten verschiedenster Listen und zum Speicher-Management.

4.10.1 MemList

Hiermit können verschiedenste Listen verwaltet werden. Basis sind ein selbstdefiniertes Array und dazu eine Instanz der Struktur „BrbMemListManagement_Typ“.

Das selbstdefinierte Array enthält dabei die Daten der Liste, die Struktur die Daten der Verwaltung.

4.10.1.1 Struktur

BrbMemListManagement_Typ		
pList	UDINT	Adresse des Array-Anfangs
nEntryLength	UDINT	Größe eines Eintrags
nIndexMax	UDINT	Maximaler Index des Arrays
nEntryCount	UDINT	Momentane Anzahl der gültigen Einträge

Vor dem ersten Aufruf einer dazugehörigen Funktion muss die Verwaltungs-Struktur mit gültigen Werten besetzt werden.

4.10.1.2 BrbMemListClear

```
signed long BrbMemListClear(struct BrbMemListManagement_Typ* pListManagement)
```

Argumente:

```
struct BrbMemListManagement_Typ* pListManagement  
Zeiger auf eine Instanz von "BrbMemListManagement_Typ"
```

Rückgabe:

```
DINT  
-1=Falsche Parameter (Nullpointer?)  
0=Kein Fehler
```

Beschreibung:

Diese Funktion löscht die gesamte Liste.

4.10.1.3 BrbMemListIn

```
signed long BrbMemListIn(struct BrbMemListManagement_Typ* pListManagement, unsigned long nIndex,  
unsigned long pNewEntry)
```

Argumente:

```
struct BrbMemListManagement_Typ* pListManagement  
Zeiger auf eine Instanz von "BrbMemListManagement_Typ"  
UDINT nIndex  
Index des Eintrags, an dem der Eintrag eingefügt werden soll  
UDINT pNewEntry  
Adresse der Daten des neuen Eintrags (muss einem Array-Eintrag entsprechen)
```

Rückgabe:

```
DINT  
-1=Falsche Parameter (Nullpointer?)  
>-1=Index, bei dem eingefügt wurde
```

Beschreibung:

Diese Funktion fügt einen Eintrag an bestimmter Stelle in der Liste ein.

4.10.1.4 BrbMemListOut

```
signed long BrbMemListOut(struct BrbMemListManagement_Typ* pListManagement, unsigned long nIndex,  
unsigned long pListEntry)
```

Argumente:

```
struct BrbMemListManagement_Typ* pListManagement  
Zeiger auf eine Instanz von "BrbMemListManagement_Typ"
```


`UDINT nIndex`
Index des Eintrags, der gelöscht werden soll
`UDINT pListEntry`
Adresse auf den die gelöschten Daten des Eintrags kopiert werden (muss einem Array-Eintrag entsprechen). Werden die Daten nicht benötigt und sollen nur gelöscht werden, dann 0.

Rückgabe:

`DINT`
-1=Falsche Parameter (Nullpointer?)
>-1=Index, bei dem gelöscht wurde

Beschreibung:

Diese Funktion gibt die Daten eines bestimmten Eintrags zurück und löscht ihn aus der Liste.

4.10.1.5 BrbMemListGetEntry

`signed long` BrbMemListGetEntry(`struct` BrbMemListManagement_Typ* pListManagement, `unsigned long` nIndex, `unsigned long` pListEntry)

Argumente:

`struct` BrbMemListManagement_Typ* pListManagement
Zeiger auf eine Instanz von "BrbMemListManagement_Typ"
`UDINT` nIndex
Index des Eintrags, der geholt werden soll
`UDINT` pListEntry
Adresse auf den die gehalten Daten des Eintrags kopiert werden (muss einem Array-Eintrag entsprechen)

Rückgabe:

`DINT`
-1=Falsche Parameter (Nullpointer?)
>-1=Index, der gefunden wurde

Beschreibung:

Diese Funktion gibt die Daten eines bestimmten Eintrags zurück, löscht ihn aber nicht aus der Liste.

4.10.2 Fifo

Die oben beschriebenen Funktionen wurden benutzt, um eine FirstIn-FirstOut-Verwaltung zu realisieren, die mit allen möglichen Arrays funktioniert.

4.10.2.1 BrbFifoIn

`signed long` BrbFifoIn(`struct` BrbMemListManagement_Typ* pListManagement, `unsigned long` pNewEntry)

Argumente:

`struct` BrbMemListManagement_Typ* pListManagement
Zeiger auf eine Instanz von "BrbMemListManagement_Typ"
`UDINT` pNewEntry
Adresse der Daten des neuen Eintrags (muss einem Array-Eintrag entsprechen)

Rückgabe:

`DINT`
-1=Falsche Parameter (Nullpointer?)
>-1=Index, bei dem eingefügt wurde

Beschreibung:

Diese Funktion fügt einen Eintrag an letzter Stelle in der Liste hinzu.

4.10.2.2 BrbFifoOut

`signed long` BrbFifoOut(`struct` BrbMemListManagement_Typ* pListManagement, `unsigned long` pListEntry)

Argumente:

`struct` BrbMemListManagement_Typ* pListManagement
Zeiger auf eine Instanz von "BrbMemListManagement_Typ"
`UDINT` pListEntry
Adresse auf den die gelöschten Daten des Eintrags kopiert werden (muss einem Array-Eintrag entsprechen). Werden die Daten nicht benötigt und sollen nur gelöscht werden, dann 0.

Rückgabe:

DINT

-1=Falsche Parameter (Nullpointer?)
>-1=Index, bei dem rausgenommen wurde

Beschreibung:

Diese Funktion gibt die Daten des ersten Eintrags zurück und löscht ihn aus der Liste.

4.10.3 Lifo

Die oben beschriebenen Funktionen wurden benutzt, um eine LastIn-FirstOut-Verwaltung zu realisieren, die mit allen möglichen Arrays funktioniert.

4.10.3.1 BrbLifoIn

```
signed long BrbLifoIn(struct BrbMemListManagement_Typ* pListManagement, unsigned long pNewEntry)
```

Argumente:

struct BrbMemListManagement_Typ* pListManagement

Zeiger auf eine Instanz von "BrbMemListManagement_Typ"

UDINT pNewEntry

Adresse der Daten des neuen Eintrags (muss einem Array-Eintrag entsprechen)

Rückgabe:

DINT

-1=Falsche Parameter (Nullpointer?)
>-1=Index, bei dem eingefügt wurde

Beschreibung:

Diese Funktion fügt einen Eintrag an letzter Stelle in der Liste hinzu.

4.10.3.2 BrbLifoOut

```
signed long BrbLifoOut(struct BrbMemListManagement_Typ* pListManagement, unsigned long pListEntry)
```

Argumente:

struct BrbMemListManagement_Typ* pListManagement

Zeiger auf eine Instanz von "BrbMemListManagement_Typ"

UDINT pListEntry

Adresse auf den die gelöschten Daten des Eintrags kopiert werden (muss einem Array-Eintrag entsprechen). Werden die Daten nicht benötigt und sollen nur gelöscht werden, dann 0.

Rückgabe:

DINT

-1=Falsche Parameter (Nullpointer?)
>-1=Index, bei dem ausgeschnitten wurde

Beschreibung:

Diese Funktion gibt die Daten des letzten Eintrags zurück und löscht ihn aus der Liste.

4.10.4 Bit-Funktionen

4.10.4.1 BrbGetBitUdint

```
plcbit BrbGetBitUdint(unsigned long nValue, unsigned short nBitNumber)
```

Argumente:

UDINT nValue

Udint-Wert

UINT nBitNumber

Nummer des Bits (0..31)

Rückgabe:

BOOL

Wert des Bits

Beschreibung:

Diese Funktion gibt das angegebene Bit aus einem Uuint-Wert zurück.

4.10.4.2 BrbSetBitUuint

```
plcbit BrbSetBitUuint(unsigned long* pValue, unsigned short nBitNumber, plcbit bBit)
```

Argumente:

`UDINT*` pValue
Zeiger auf den Uuint-Wert
`UINT` nBitNumber
Nummer des Bits (0..31)
`BOOL` bBit
0=Löschen, 1=Setzen

Rückgabe:

`BOOL`
Wert des Bits vor dem Setzen

Beschreibung:

Diese Funktion setzt oder löscht das angegebene Bit in einem Uuint-Wert.

4.10.4.3 BrbGetBitMaskUuint

```
plcbit BrbGetBitMaskUuint(unsigned long nValue, unsigned long nBitMask)
```

Argumente:

`UDINT` nValue
Uuint-Wert
`UDINT` nBitMask
Wert der Bitmaske

Rückgabe:

`BOOL`
Bitmaske enthalten

Beschreibung:

Diese Funktion gibt zurück, ob die Bitmaske im Uuint-Wert enthalten ist.

4.10.4.4 BrbSetBitMaskUuint

```
plcbit BrbSetBitMaskUuint(unsigned long* pValue, unsigned long nBitMask, plcbit bSet)
```

Argumente:

`UDINT*` pValue
Zeiger auf den Uuint-Wert
`UDINT` nBitMask
Wert der Bitmaske
`BOOL` bSet
0=Löschen, 1=Setzen

Rückgabe:

`BOOL`
Bitmaske war vor dem Setzen enthalten

Beschreibung:

Diese Funktion setzt oder löscht die angegebene Bitmaske in einem Uuint-Wert.

4.10.4.5 BrbGetBitUuint

```
plcbit BrbGetBitUuint(unsigned short nValue, unsigned short nBitNumber)
```

Argumente:

`UINT` nValue
Uuint-Wert
`UINT` nBitNumber
Nummer des Bits (0..15)

Rückgabe:

BOOL

Wert des Bits

Beschreibung:

Diese Funktion gibt das angegebene Bit aus einem Uint-Wert zurück.

4.10.4.6 BrbSetBitUint

```
plcbit BrbSetBitUint(unsigned short* pValue, unsigned short nBitNumber, plcbit bBit)
```

Argumente:

UINT* pValue

Zeiger auf den Uint-Wert

UINT nBitNumber

Nummer des Bits (0..15)

BOOL bBit

0=Löschen, 1=Setzen

Rückgabe:

BOOL

Wert des Bits vor dem Setzen

Beschreibung:

Diese Funktion setzt oder löscht das angegebene Bit in einem Uint-Wert.

4.10.4.7 BrbGetBitMaskUint

```
plcbit BrbGetBitMaskUint(unsigned short nValue, unsigned short nBitMask)
```

Argumente:

UINT nValue

Udint-Wert

UINT nBitMask

Wert der Bitmaske

Rückgabe:

BOOL

Bitmaske enthalten

Beschreibung:

Diese Funktion gibt zurück, ob die Bitmaske im Uint-Wert enthalten ist.

4.10.4.8 BrbSetBitMaskUint

```
plcbit BrbSetBitMaskUint(unsigned short* pValue, unsigned short nBitMask, plcbit bSet)
```

Argumente:

UINT* pValue

Zeiger auf den Uuint-Wert

UINT nBitMask

Wert der Bitmaske

BOOL bSet

0=Löschen, 1=Setzen

Rückgabe:

BOOL

Bitmaske war vor dem Setzen enthalten

Beschreibung:

Diese Funktion setzt oder löscht die angegebene Bitmaske in einem Usint-Wert.

4.10.4.9 BrbGetBitUsint

```
plcbit BrbGetBitUsint(unsigned char nValue, unsigned short nBitNumber)
```

Argumente:

USINT nValue

Usint-Wert

UINT nBitNumber

Nummer des Bits (0..7)

Rückgabe:

BOOL

Wert des Bits

Beschreibung:

Diese Funktion gibt das angegebene Bit aus einem Usint-Wert zurück.

4.10.4.10 BrbSetBitUsint

```
plcbit BrbSetBitUsint(unsigned char* pValue, unsigned short nBitNumber, plcbit bBit)
```

Argumente:

USINT* pValue

Zeiger auf den Usint-Wert

UINT nBitNumber

Nummer des Bits (0..7)

BOOL bBit

0=Löschen, 1=Setzen

Rückgabe:

BOOL

Wert des Bits vor dem Setzen

Beschreibung:

Diese Funktion setzt oder löscht das angegebene Bit in einem Usint-Wert.

4.10.4.11 BrbGetBitMaskUsint

```
plcbit BrbGetBitMaskUsint(unsigned char nValue, unsigned char nBitMask)
```

Argumente:

USINT nValue

Udint-Wert

USINT nBitMask

Wert der Bitmaske

Rückgabe:

BOOL

Bitmaske enthalten

Beschreibung:

Diese Funktion gibt zurück, ob die Bitmaske im Usint-Wert enthalten ist.

4.10.4.12 BrbSetBitMaskUsint

```
plcbit BrbSetBitMaskUsint(unsigned char* pValue, unsigned char nBitMask, plcbit bSet)
```

Argumente:

USINT* pValue

Zeiger auf den Udint-Wert

USINT nBitMask

Wert der Bitmaske

BOOL bSet

0=Löschen, 1=Setzen

Rückgabe:

BOOL

Bitmaske war vor dem Setzen enthalten

Beschreibung:

Diese Funktion setzt oder löscht die angegebene Bitmaske in einem Usint-Wert.

4.10.5 ByteArray-Funktionen

Manchmal ist es hilfreich, boolesche Informationen bitcodiert in einem größeren Datentyp zu speichern. Dies verringert den Speicherbedarf gegenüber einem BOOL-Array drastisch (1 BOOL benötigt 1 Byte). Der größte dafür verwendbare Standard-Datentyp ist UDINT, welcher 32 Bits zur Verfügung stellt.

Braucht man mehr als 32 Bits, ist die Umsetzung komplexer.

Diese Funktionen unterstützen hierbei. Als Speicher dient ein vom Anwender definiertes USINT-Array.

Die jeweils anzugebende Bitnummer zeigt auf ein bestimmtes Bit im Array:

Byte-Index	Bitnummern des Bytes	Anzugebende Bitnummer
0	0..7	0..7
1	0..7	8..15
2	0..7	16..23
3	0..7	24..31
4	0..7	32..39
5	0..7	40..47
...	0..7	...

So können problemlos sehr viele boolesche Daten speichersparend gehalten werden.

4.10.5.1 BrbGetByteArrayBit

```
plcbit BrbGetByteArrayBit(unsigned long pByteArray, unsigned long nIndexMax, unsigned long nBit-  
Number)
```

Argumente:

UDINT pByteArray
 Zeiger auf das Byte-Array
UDINT nIndexMax
 Maximaler Index des Byte-Arrays
UDINT nBitNumber
 Nummer des Bits (0..n)

Rückgabe:

BOOL
 Wert des Bits

Beschreibung:

Diese Funktion gibt das angegebene Bit aus einem Byte-Array zurück.

Ist die Bitnummer nicht mehr im Array enthalten, wird 0 zurückgegeben.

4.10.5.2 BrbSetByteArrayBit

```
plcbit BrbSetByteArrayBit(unsigned long pByteArray, unsigned long nIndexMax, unsigned long nBit-  
Number, plcbit bBit)
```

Argumente:

UDINT pByteArray
 Zeiger auf das Byte-Array
UDINT nIndexMax
 Maximaler Index des Byte-Arrays
UDINT nBitNumber
 Nummer des Bits (0..n)
BOOL bBit
 Zu setzender Wert

Rückgabe:

BOOL
 Vorheriger Wert des Bits

Beschreibung:

Diese Funktion setzt oder löscht das angegebene Bit in einem Byte-Array.

Ist die Bitnummer nicht mehr im Array enthalten, wird der Wert natürlich nicht gesetzt.

4.10.5.3 BrbSetByteArrayBits

```
plcbit BrbSetByteArrayBits(unsigned long pByteArray, unsigned long nIndexMax, unsigned long nBit-  
NumberFrom, unsigned long nBitNumberTo, plcbit bBit)
```

Argumente:

`UDINT` pByteArray
 Zeiger auf das Byte-Array
`UDINT` nIndexMax
 Maximaler Index des Byte-Arrays
`UDINT` nBitNumberFrom
 Nummer des Start-Bits (0..n)
`UDINT` nBitNumberTo
 Nummer des End-Bits (0..n)
`BOOL` bBit
 Zu setzender Wert für alle Bits

Rückgabe:

`BOOL`
 Zu setzender Wert

Beschreibung:

Diese Funktion setzt oder löscht die Bits im angegebenen Bereich in einem Byte-Array.
Ist eine Bitnummer des Bereichs nicht mehr im Array enthalten, wird die Schleife abgebrochen.
Hinweis: Ist die End-Nummer kleiner als die Start-Nummer, wird der Bereich trotzdem bearbeitet.

4.11 Math

In diesem Paket finden sich hilfreiche mathematische Funktionen.

4.11.1 BrbAbsReal

```
float BrbAbsReal(float rValue)
```

Argumente:

REAL rValue
Wert

Rückgabe:

REAL
Absoluter Wert

Beschreibung:

Diese Funktion gibt den absoluten Wert, also immer den positiven Wert einer Zahl zurück.
In den IEC-Sprachen gibt es dazu die Bibliothek „OPERATOR“, welche in ANSI-C aber nicht eingesetzt werden darf. Die C-interne Funktion „abs()“ erlaubt nur DINT als Eingang.

4.11.2 BrbAbsLReal

```
double BrbAbsLReal(double rValue)
```

Beschreibung:

Diese Funktion verhält sich wie 'BrbAbsReal' (siehe oben), erwartet aber den Datentyp 'LREAL'.

4.11.3 BrbIsNearlyZeroReal

```
plcbit BrbIsNearlyZeroReal(float rValue, float rTolerance)
```

Argumente:

REAL rValue
Zu überprüfender Wert
REAL rTolerance
Grenz-Toleranz

Rückgabe:

BOOL
0= Wert kann nicht als 0 interpretiert werden
1= Wert kann als 0 interpretiert werden

Beschreibung:

Bei Berechnungen ergeben sich manchmal sehr kleine Werte, die nahe an 0 liegen (z.B. 1e-009).
Diese Funktion gibt zurück, ob sich ein Wert als 0 interpretieren lässt.
Die Toleranz wird von 0.0 subtrahiert und zu 0.0 addiert (eine negative Toleranz wird intern ins Positive korrigiert) und dadurch Grenzwerte definiert.
Ist der Wert innerhalb der inklusiven Grenzen, wird 'TRUE' zurückgegeben.

4.11.4 BrbIsNearlyZeroLReal

```
plcbit BrbIsNearlyZeroLReal(double rValue, double rTolerance)
```

Beschreibung:

Diese Funktion verhält sich wie 'BrbIsNearlyZeroReal' (siehe oben), erwartet aber den Datentyp 'LREAL'.

4.11.5 BrbCompareReal

```
signed short BrbCompareReal(float rValue1, float rValue2)
```


Argumente:

`REAL` rValue1
1.Wert
`REAL` rValue2
2.Wert

Rückgabe:

`INT`
-1= Der 1.Wert ist kleiner als der 2.
0= Die Werte sind gleich
1= Der 1.Wert ist größer als der 2.
-32768 (nBRB_INT_MIN) = Der 1.Wert ist NAN (Not a number = Keine Zahl) oder INF (Infinity = Unendlich)
32767 (nBRB_INT_MAX) = Der 2.Wert ist NAN (Not a number = Keine Zahl) oder INF (Infinity = Unendlich)

Beschreibung:

Die Funktion vergleicht zwei REAL-Werte. NAN oder INF werden dabei berücksichtigt.

4.11.6 BrbCompareLReal

```
signed short BrbCompareLReal(double rValue1, double rValue2)
```

Beschreibung:

Diese Funktion verhält sich wie 'BrbCompareReal' (siehe oben), erwartet aber den Datentyp 'LREAL'.

4.11.7 BrbIsWithinRangeDint

```
plcbit BrbIsWithinRangeDint(signed long nValue, signed long nRangeMin, signed long nRangeMax,  
signed long nTolerance)
```

Argumente:

`DINT` nValue
Zu überprüfender Wert
`DINT` nRangeMin
Untere Grenze
`DINT` nRangeMax
Obere Grenze
`DINT` nTolerance
Grenz-Toleranz

Rückgabe:

`BOOL`
0= Wert ist außerhalb des Bereichs
1= Wert ist innerhalb des Bereichs

Beschreibung:

Diese Funktion gibt zurück, ob sich ein Wert innerhalb eines Bereichs befindet.
Ist der Eingang 'nRangeMin' größer als 'nRangeMax', wird dies intern korrigiert.
Die Toleranz wird bei der unteren Grenze subtrahiert, bei der oberen Grenze addiert (eine negative Toleranz wird intern ins Positive korrigiert).
Ist der Wert innerhalb der inklusiven Grenzen, wird 'TRUE' zurückgegeben.

4.11.8 BrbIsWithinRangeUdint

```
plcbit BrbIsWithinRangeUdint(unsigned long nValue, unsigned long nRangeMin, unsigned long nRange-  
Max, unsigned long nTolerance)
```

Beschreibung:

Diese Funktion verhält sich wie 'BrbIsWithinRangeDint' (siehe oben), erwartet aber den Datentyp 'UDINT'.

4.11.9 BrbIsWithinRangeReal

```
plcbit BrbIsWithinRangeReal(float rValue, float rRangeMin, float rRangeMax, float rTolerance)
```

Beschreibung:

Diese Funktion verhält sich wie 'BrbIsWithinRangeDint' (siehe oben), erwartet aber den Datentyp 'REAL'.

4.11.10 BrbIsWithinRangeLReal

```
plcbit BrbIsWithinRangeLReal(double rValue, double rRangeMin, double rRangeMax, double rTolerance)
```

Beschreibung:

Diese Funktion verhält sich wie 'BrbIsWithinRangeDint' (siehe oben), erwartet aber den Datentyp 'LREAL'.

4.11.11 BrbGetAngleRad

```
float BrbGetAngleRad(float rAngleDeg)
```

Argumente:

REAL rAngleDeg
Winkel in Grad

Rückgabe:

REAL
Winkel im Bogenmaß

Beschreibung:

Mathematische Funktionen, welche eine Winkel-Angabe benötigen, z.B. sin, cos und tan, erwarten diesen Winkel im Bogenmaß. Diese Funktion rechnet einen Winkel von Grad ins Bogenmaß um.

4.11.12 BrbGetAngleDeg

```
float BrbGetAngleDeg(float rAngleRad)
```

Argumente:

REAL rAngleRad
Winkel im Bogenmaß

Rückgabe:

REAL
Winkel in Grad

Beschreibung:

Mathematische Funktionen, welche einen Winkel zurückgeben, z.B. sinh, cosh und tanh, geben diesen im Bogenmaß zurück. Diese Funktion rechnet einen Winkel vom Bogenmaß in Grad um.

4.11.13 BrbNormalizeAngleRad

```
float BrbNormalizeAngleRad(float rAngleRad)
```

Argumente:

REAL rAngleRad
Winkel im Bogenmaß

Rückgabe:

REAL
Normalisierter Winkel im Bogenmaß

Beschreibung:

Diese Funktion verschiebt einen Winkel außerhalb des Bereichs in den Bereich $0..2\pi$. Damit kann der Überlauf von Winkeln korrigiert werden.

4.11.14 BrbNormalizeAngleDeg

```
BrbNormalizeAngleDeg(float rAngleDeg, plcbit bKeep360)
```

Argumente:

REAL rAngleDeg
Winkel in Grad
BOOL bKeep360
1= 360° nicht in 0° wandeln

Rückgabe:

REAL
Normalisierter Winkel in Grad

Beschreibung:

Diese Funktion verschiebt einen Winkel außerhalb des Bereichs in den Bereich 0..360°. Damit kann der Überlauf von Winkeln korrigiert werden. In manchen Fällen sollten genau 360.00° nicht in 0.00° gewandelt werden. Dazu gibt es einen eigenen Eingang.

4.11.15 BrbIsWithinRangeAngle

```
plcbit BrbIsWithinRangeAngle(double rAngleAct, double rAngleStart, double rAngleEnd)
```

Argumente:

LREAL rAngleAct
Zu prüfender Winkel in [°]
LREAL rAngleStart
Start-Winkel des Bereichs [0° <= rAngleStart < 360°]
LREAL rAngleEnd
End-Winkel des Bereichs [0° <= rAngleEnd < 360°]

Rückgabe:

BOOL
0= Wert ist außerhalb des Bereichs
1= Wert ist innerhalb des Bereichs

Beschreibung:

Diese Funktion gibt zurück, ob sich ein Winkel innerhalb eines Bereichs befindet. Dabei wird der mögliche Überlauf zwischen 360° und 0° berücksichtigt.
Ist der Winkel innerhalb der inklusiven Grenzen, wird ‚TRUE‘ zurückgegeben.

4.11.16 BrbDetectAngleTransition

```
plcbit BrbDetectAngleTransition(double rTransPosition, plcbit bDirection, double rPosAct, double rPosOld)
```

Argumente:

LREAL rTransPosition
Zu erkennende Übergangs-Position in [°]
BOOL bDirection
Richtung: 0=Negativ, 1 =Positiv
LREAL rPosAct
Aktuelle Rundachs-Position [0° <= rPosAct < 360°]
LREAL rPosOld
Alte Rundachs-Position [0° <= rPosOld < 360°] vom vorigen Taskzyklus

Rückgabe:

BOOL
1=Übergang erkannt

Beschreibung:

Diese Funktion erkennt den Übergang an einer bestimmten Winkel-Position einer 360°-Rundachse aus einer angegebenen Richtung. Dabei wird der mögliche Überlauf zwischen 360° und 0° aus beiden Richtungen berücksichtigt. So kann z.B. beim Überfahren einer Winkel-Position eine Aktion ausgelöst werden.

Zum Erkennen des Übergangs und der Richtung wird sowohl der alte als auch der aktuelle Positions-Wert benötigt. Das bedeutet, dass nach dem Aufruf der Funktion der alte Positions-Wert gespei-

chert und im nächsten Zyklus wieder übergeben werden muss. Müssen mehrere Positions-Übergänge erkannt werden, so muss der alte Positions-Wert nur einmal gespeichert werden, indem er erst nach allen Aufrufen aktualisiert wird.

Beispiel:

```
// Erkennen des Vorwärts-Übergangs auf 0°
bDetect000 = BrbDetectAngleTransition (0.0, 1, Axis.rPosAct, rPosOld);
// Erkennen des Vorwärts-Übergangs auf 90°
bDetect090 = BrbDetectAngleTransition (90.0, 1, Axis.rPosAct, rPosOld);
// Erkennen des Vorwärts-Übergangs auf 180°
bDetect180 = BrbDetectAngleTransition (180.0, 1, Axis.rPosAct, rPosOld);
// Erkennen des Vorwärts-Übergangs auf 270°
bDetect270 = BrbDetectAngleTransition (270.0, 1, Axis.rPosAct, rPosOld);
// Erkennen des Rückwärts-Übergangs auf 135°
bDetect135 = BrbDetectAngleTransition (135.0, 0, Axis.rPosAct, rPosOld);
rPosOld = Axis.rPosAct; // Merken der Position für den nächsten Task-Zyklus
```

Hinweis: Die Funktion arbeitet nur korrekt, wenn der Positions-Sprung pro Task-Zyklus unter 90° liegt. In der Praxis ist es jedoch sehr unwahrscheinlich, dass diese Bedingung nicht eingehalten wird.

4.11.17 BrbGetDistance2d

```
double BrbGetDistance2d(double rX1, double rY1, double rX2, double rY2)
```

Argumente:

LREAL rX1
X-Koordinate des 1. Punkts
LREAL rY1
Y-Koordinate des 1. Punkts
LREAL rX2
X-Koordinate des 2. Punkts
LREAL rY2
Y-Koordinate des 2. Punkts

Rückgabe:

LREAL
Distanz zwischen den zwei Punkten

Beschreibung:

Diese Funktion berechnet die Distanz von zwei Punkten im 2-dimensionalen Raum. Damit kann z.B. das Annähern an eine Achs-Position einer X/Y-Achs-Kombination aus jeder Richtung ermittelt werden, um vor dem Erreichen eine Aktion zu starten.

4.11.18 BrbRoundDint

```
signed long BrbRoundDint(signed long nValue, enum BrbRound_ENUM eRound, unsigned char nDigits)
```

Argumente:

DINT nValue
Zu rundender Wert
BrbRound_ENUM eRound
Rundungs-Typ

BrbRound_ENUM	
eBRB_ROUND_APPROP	Rundung gemäss letzter Stelle(n). Grenze bei 5
eBRB_ROUND_UP	Aufrundung
eBRB_ROUND_DOWN	Abrundung

USINT nDigits
Stelle, auf die gerundet wird

Rückgabe:

DINT
Der gerundete Wert

Beschreibung:

Rundet einen Wert nach Vorgaben.
Angabe der Stelle (mind. 1):

1 1-Stelle
2 10-Stelle
3 100-Stelle
...

Der Rundungstyp gibt an, welche Rundung vorgenommen wird: Immer Aufrundung, immer Abrundung oder gemäß der letzten Stelle(n). Die Grenze ist dabei immer die 5 an der Rundungsstelle.

Beispiele:

Angabe	Ergebnis
1245, App, 1	1250
1244, App, 1	1240
1249, App, 2	1200
1250, App, 2	1300
1200, Up, 3	2000
1200, Down, 3	1000
1200, Up, 4	10000
1200, Down, 4	0

4.11.19 BrbScaleLReal

```
double BrbScaleLReal(double rRaw, double rRawMin, double rRawMax, double rScaledMin, double  
rScaledMax, double rFactor, double rOffset)
```

Argumente:

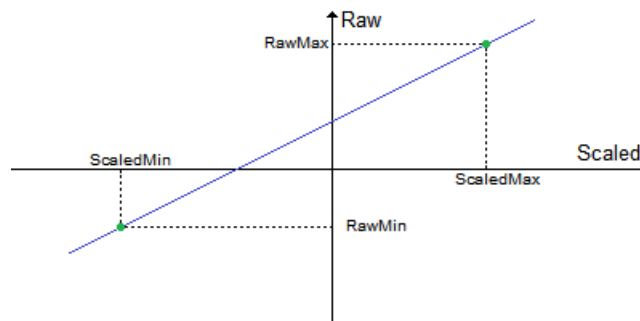
`LREAL` rRaw
Rohwert
`LREAL` rRawMin
Minimaler Rohwert
`LREAL` rRawMax
Maximaler Rohwert
`LREAL` rScaledMin
Minimaler Skalierwert
`LREAL` rScaledMax
Maximaler Skalierwert
`LREAL` rFactor
Umrechnungs-Faktor
`LREAL` rOffset
Umrechnungs-Offset

Rückgabe:

`REAL`
Skalierter Wert

Beschreibung:

Rechnet einen analogen Rohwert mithilfe einer linearen Kennlinie auf einen skalierten Wert um.



Die Kennlinie wird mit den Eingangs-Koordinaten bestimmt.

Mit dem Eingang „rFactor“ kann zusätzlich die Steigung verändert werden. Ist das nicht gewollt, so muss „1.0“ gesetzt werden.

Mit dem Eingang „rOffset“ kann die Kennlinie nach rechts oder links verschoben werden. Ist das nicht gewollt, so muss „0.0“ gesetzt werden.

4.11.20 BrbScaleAnalogInput

```
float BrbScaleAnalogInput(signed short nRaw, signed short nRawMin, signed short nRawMax, float rScaledMin, float rScaledMax, float rFactor, float rOffset)
```

Argumente:

INT nRaw
Rohwert von der Analog-Eingangs-Karte

INT nRawMin
Minimaler Rohwert

INT nRawMax
Maximaler Rohwert

REAL rScaledMin
Minimaler Einheitswert

REAL rScaledMax
Maximaler Einheitswert

REAL rFactor
Umrechnungs-Faktor

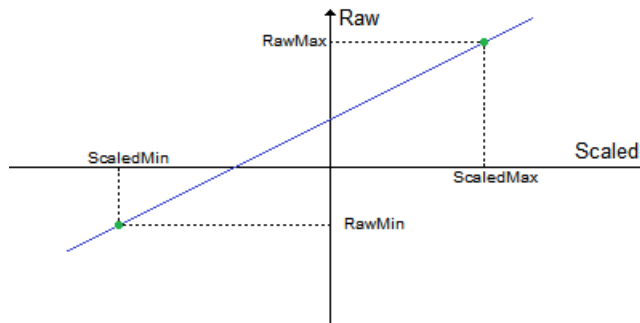
REAL rOffset
Umrechnungs-Offset

Rückgabe:

REAL
Skalierter Einheitswert

Beschreibung:

Rechnet einen analogen Rohwert (z.B. -32767..+32767) mithilfe einer linearen Kennlinie auf einen Einheitswert (z.B. -100..+100°C) um.



Die Kennlinie wird mit den Eingangs-Koordinaten bestimmt.

Mit dem Eingang „rFactor“ kann zusätzlich die Steigung verändert werden. Ist das nicht gewollt, so muss „1.0“ gesetzt werden.

Mit dem Eingang „rOffset“ kann die Kennlinie nach rechts oder links verschoben werden. Ist das nicht gewollt, so muss „0.0“ gesetzt werden.

4.11.21 BrbScaleAnalogOutput

```
signed short BrbScaleAnalogOutput(float rScaled, signed short nRawMin, signed short nRawMax, float rScaledMin, float rScaledMax, float rFactor, float rOffset)
```

Argumente:

REAL rScaled
Einheitswert

INT nRawMin
Minimaler Rohwert

INT nRawMax
Maximaler Rohwert

REAL rScaledMin
Minimaler Einheitswert

REAL rScaledMax
Maximaler Einheitswert
REAL rFactor
Umrechnungs-Faktor
REAL rOffset
Umrechnungs-Offset

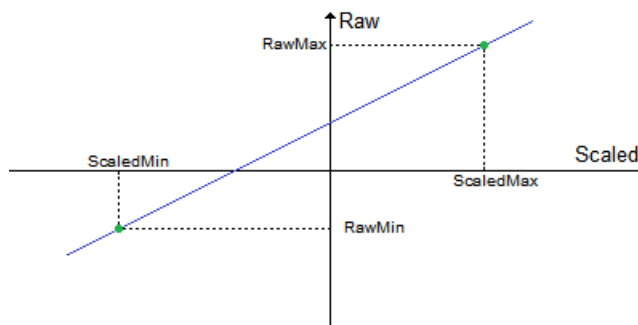
Rückgabe:

INT

Rohwert für die Analog-Ausgangs-Karte

Beschreibung:

Rechnet einen Einheitswert (z.B. -100..+100°C) mithilfe einer linearen Kennlinie auf einen analogen Rohwert (z.B. -32767..+32767) um.



Die Kennlinie wird mit den Eingangs-Koordinaten bestimmt.

Mit dem Eingang „rFactor“ kann zusätzlich die Steigung verändert werden. Ist das nicht gewollt, so muss „1.0“ gesetzt werden.

Mit dem Eingang „rOffset“ kann die Kennlinie nach rechts oder links verschoben werden. Ist das nicht gewollt, so muss „0.0“ gesetzt werden.

Achtung: Bis zur Version 5.00 skalierte diese Funktion nicht richtig, weil die Kennlinie falsch berechnet wurde.

Die Skalierung wurde in V5.00 komplett neu implementiert und verhält sich jetzt äquivalent zu „BrbScaleAnalogInput“. Außerdem wird jetzt der Ausgangswert auf ‚nRawMin‘ bzw. ‚nRawMax‘ begrenzt, so dass keine ungültigen Werte an die Ausgangskarte kommen können bzw. kein Überlauf stattfinden kann.

Äquivalent bedeutet, dass mit derselben Kennlinie (also auch denselben Parametern ‚rFactor‘ und ‚rOffset‘) mithilfe von ‚BrbScaleAnalogInput‘ und ‚BrbScaleAnalogOutput‘ gleichwertig hin und her gerechnet werden kann.

Beispiel:

```
BrbScaleAnalogOutput (34.56 , -30000, +30000, -100.0, +100.0, 1.5, 12.3) = 4452
BrbScaleAnalogInput  (4452 , -30000, +30000, -100.0, +100.0, 1.5, 12.3) = 34.56
```

Hinweis: Aufgrund Ungenauigkeiten bei der Fließkomma-Rechnung können kleine Abweichungen im Nachkomma-Bereich entstehen.

Achtung: Zwar ist die Signatur gleichgeblieben, die Logik dieser Funktion ist aber nicht mehr kompatibel zur alten Version. Beim Update auf die Version 5.00 oder höher sollte deshalb die Verwendung dieser Funktion geprüft werden.

4.12 Random

In diesem Paket finden sich Funktionen zum Erzeugen von Zufallswerten.

4.12.1 BrbGetRandomPercent

```
float BrbGetRandomPercent()
```

Argumente:

Rückgabe:

REAL

Die Zufallszahl

Beschreibung:

Erzeugt eine Zufallszahl zwischen inklusive 0.0 und 1.0.

4.12.2 BrbGetRandomBool

```
plcbit BrbGetRandomBool()
```

Argumente:

Rückgabe:

BOOL

Der Zufallswert

Beschreibung:

Erzeugt einen Zufallswert mit 0 (=False) oder 1 (=True).

4.12.3 BrbGetRandomUdint

```
signed long BrbGetRandomUdint(unsigned long nMin, unsigned long nMax)
```

Argumente:

UDINT nMin

Untere Grenze der Zufallszahl

UDINT nMax

Obere Grenze der Zufallszahl

Rückgabe:

UDINT

Die Zufallszahl

Beschreibung:

Erzeugt eine Zufallszahl zwischen inklusive nMin und nMax.

4.12.4 BrbGetRandomDint

```
signed long BrbGetRandomDint(signed long nMin, signed long nMax)
```

Argumente:

DINT nMin

Untere Grenze der Zufallszahl

DINT nMax

Obere Grenze der Zufallszahl

Rückgabe:

DINT

Die Zufallszahl

Beschreibung:

Erzeugt eine Zufallszahl zwischen inklusive nMin und nMax.

4.12.5 BrbGetRandomText

```
unsigned long BrbGetRandomText(plcstring* pText, unsigned long nTextSize, unsigned long nTextLength)
```

Argumente:

STRING* pText
Zeiger auf den String, der erzeugten Text aufnehmen soll
UDINT nTextSize
Größe der String-Variablen
UDINT nTextLength
Länge des zu erzeugenden Strings

Rückgabe:

UDINT
Länge des tatsächlich erzeugten Texts

Beschreibung:

Erzeugt einen Zufallstext mit bestimmter Länge. Enthalten sind nur Zahlen, Großbuchstaben und Kleinbuchstaben.

4.12.6 BrbGetRandomString

```
unsigned long BrbGetRandomString(plcstring* pText, unsigned long nTextSize, unsigned long nTextLength, enum BrbRandomString_ENUM eSelection)
```

Argumente:

STRING* pText
Zeiger auf den String, der erzeugten Text aufnehmen soll
UDINT nTextSize
Größe der String-Variablen
UDINT nTextLength
Länge des zu erzeugenden Strings
BrbRandomString_ENUM eSelection
Bitweise Codierung, welcher Zeichen-Block enthalten sein soll

BrbRandomString_ENUM			
eBRB_RANDOM_STRING_NUMBERS	1	Zahler	
eBRB_RANDOM_STRING_UPPER_LETTERS	2	Großbuchstaben	
eBRB_RANDOM_STRING_LOWER_LETTERS	4	Kleinbuchstaben	
eBRB_RANDOM_STRING_SPACE	8	Leerzeichen	
eBRB_RANDOM_STRING_PUNCTUATION	16	Satzzeichen	
eBRB_RANDOM_STRING_SYMBOLS	32	Symbole	
eBRB_RANDOM_STRING_CONTROLCHARS	64	Steuerzeichen	

Rückgabe:

UDINT
Länge des tatsächlich erzeugten Texts

Beschreibung:

Erzeugt einen Zufallstext mit bestimmter Länge. Die enthaltenen Zeichen können blockweise ausgewählt werden.

Auswahl	ASCII-Code (von/bis)	Zeichen
eBRB_RANDOM_STRING_NUMBERS	48..57	0..9
eBRB_RANDOM_STRING_UPPER_LETTERS	65..90	A..Z
eBRB_RANDOM_STRING_LOWER_LETTERS	97..122	a..z
eBRB_RANDOM_STRING_SPACE	32	" "
eBRB_RANDOM_STRING_PUNCTUATION	33..47	!"#\$%&'()*+,-./
	58..64	::<=>?@
	91..96	[] ^ _ `
	123..126	{ } ~
eBRB_RANDOM_STRING_SYMBOLS	128..255	
eBRB_RANDOM_STRING_CONTROLCHARS	1..31;127	

Die Blockauswahl kann einfach addiert werden:

```
eSelection = eBRB_RANDOM_STRING_UPPER_LETTERS + eBRB_RANDOM_STRING_SPACE
```

erzeugt einen Text, in dem nur Großbuchstaben und Leerzeichen enthalten sind.

4.12.7 BrbGetRandomStringExt

```
unsigned long BrbGetRandomStringExt(plcstring* pText, unsigned long nTextSize, unsigned long  
nTextLength, plcstring* pSelection)
```

Argumente:

`STRING*` pText
Zeiger auf den String, der erzeugten Text aufnehmen soll

`UDINT` nTextSize
Größe der String-Variablen

`UDINT` nTextLength
Länge des zu erzeugenden Strings

`STRING*` pSelection
Zeiger auf den String, welcher die zu verwendeten Zeichen enthält

Rückgabe:

`UDINT`
Länge des tatsächlich erzeugten Texts

Beschreibung:

Erzeugt einen Zufallstext mit bestimmter Länge. Die Auswahl an Zeichen, welche enthalten sein dürfen, wird als eigener String übergeben.

Der Auswahl-String muss wie jeder andere Text auch mit dem ASCII-Code 0 abgeschlossen sein.

In der Auswahl können alle ASCII-Codes von 1 bis 255 (also auch Steuerzeichen) enthalten sein.

Wenn ein Zeichen öfter verwendet werden soll als andere, muss es nur öfter in der Auswahl vorkommen.

4.13 IpAndSubnet

In diesem Paket finden sich Funktionen zum Prüfen und Korrigieren von Ip-Adressen und Subnet-Masken.

Für diese Angaben gibt es verschiedene Regeln zu beachten. Wenn nun eine oder mehrere dieser Angaben vom Benutzer an der Visu eingegeben werden, kann überprüft werden, ob sie gültig sind, bevor sie ans Betriebssystem weitergegeben werden. Bei manchen Funktionen wird die entsprechende Angabe auch gleich korrigiert.

4.13.1 BrbIpToNumericalArray

```
unsigned short BrbIpToNumericalArray(plcstring* pIp, unsigned long pUsintArray)
```

Argumente:

`STRING*` pIpAddress
Zeiger auf die Ip-Adresse
`UDINT` pUsintArray
Zeiger auf ein USINT-Array mit 4 Elementen

Rückgabe:

`UINT`
eBRB_ERR_OK = 0
eBRB_ERR_NULL_POINTER = 50000

Beschreibung:

Diese Funktion trägt die einzelnen Segmente einer textuellen Ip-Adresse im Format ‚xxx.xxx.xxx.xxx‘ in ein übergebenes USINT-Array ein und wandelt sie somit in ein numerisches Format, welches manchmal zur Weiterverarbeitung sehr nützlich ist. Sie wird intern von anderen Funktionen verwendet.

Achtung: Das als Zeiger übergebene Array muss vom Typ USINT sein und 4 Elemente haben. Ansonsten kann es zu Speicherungsverletzungen führen!

4.13.2 BrbCheckIpAddress

```
plcbrit BrbCheckIpAddress(plcstring* pIpAddress)
```

Argumente:

`STRING*` pIpAddress
Zeiger auf die Ip-Adresse

Rückgabe:

`BOOL`
0

Beschreibung:

Diese Funktion prüft und berichtigt eine textuelle Ip-Adresse oder eine Subnet-Maske im Format ‚xxx.xxx.xxx.xxx‘ auf folgende Punkte:

- Es müssen 4 Segmente angegeben sein, welche durch „.“ getrennt sind.
- In den Segmenten dürfen nur Zahlen vorkommen
- Jedes Segment wird auf den Bereich 0...255 limitiert
- Ein Segment darf keine führenden Nullen enthalten

Führende Nullen eines Segments werden vom Betriebssystem als Oktal-Präfix erkannt und somit das Segment als Oktal-Zahl ausgewertet. Weil dies eigentlich nie beabsichtigt ist, werden führende Nullen entfernt.

Entspricht ein Segment nicht den sonstigen Prüfungen, wird es mit „0“ ersetzt.

4.13.3 BrbCheckSubnet

```
plcbrit BrbCheckSubnet(plcstring* pSubnet)
```

Argumente:

`STRING*` pSubnet

Zeiger auf die Subnet-Maske

Rückgabe:

BOOL

0= Die Subnet-Maske ist nicht gültig

1= Die Subnet-Maske ist gültig

Beschreibung:

Diese Funktion prüft, ob eine textuelle Subnet-Maske im Format ‚xxx.xxx.xxx.xxx‘ gültige Werte enthält. So kann z.B. die Eingabe eines Benutzers geprüft werden.

Hinweis: Es empfiehlt sich, das Subnet nach Eingabe, aber vor dem Aufruf dieser Funktion durch die Funktion `BrbCheckIpAddress` (siehe oben) berichtigen zu lassen.

Hintergrund:

Eine Ip-Adresse enthält immer 2 Informationen: Die Adressierung des Netzwerks, in dem der Teilnehmer untergebracht ist, und die Adresse, unter der der Teilnehmer innerhalb dieses Netzwerks angesprochen wird. Die Subnet-Maske dient nun dazu, diese beiden Infos aus der gesamten Teilnehmer-Adresse extrahieren zu können. Alle Bits, die in der Subnet-Maske auf 1 stehen, werden zur Kennung des Netzwerkes verwendet (Netzwerk-Anteil). Alle Bits, die in der Subnet-Maske auf 0 stehen, werden zur Kennung des Teilnehmers verwendet (Host-Anteil).

Beispiel Subnet-Maske 255.255.255.0

Die ersten drei Segmente (also die ersten $3 \times 8 = 24$ Bits) einer Ip-Adresse bezeichnen das Netzwerk, das letzte Segment (also die letzten 8 Bits) werden zur Unterscheidung der Teilnehmer verwendet.

Mit 8 Bits lassen sich die Zahlen von $2^8 = 0$ bis 255 darstellen, d.h., in diesem Netzwerk können sich 256 verschiedene Teilnehmer befinden. Werden mehr Teilnehmer innerhalb eines Netzwerkes benötigt, so kann die Subnet-Maske auch eine andere Grenze vorgeben.

Beispiel Subnet-Maske 255.255.252.0

Hier werden nur die ersten 22 Bits fürs die Netzwerk-Adresse verwendet und die restlichen 10 für die Teilnehmer. Es sind also $2^{10} = 1024$ Teilnehmer möglich.

Diese Aufteilung bedingt aber, dass ab dem ersten Bit, das auf 0 gestellt ist, nur Bits folgen, die ebenfalls 0 sind. Eine Subnet-Maske von z.B. 255.255.254.2 ist also ungültig, weil ein Bit im 3 Segment auf 0 steht, im 4. Segment aber wieder Bits mit 1 folgen. Ebenso ungültig ist es, keinen Netzwerk-Anteil (0.0.0.0) oder keinen Host-Anteil (255.255.255.255) zu definieren.

Alles dies wird von der Funktion geprüft.

4.13.4 BrbGetIpAddressType

```
enum BrbIpAdrType_ENUM BrbGetIpAddressType(plcstring* pIp, plcstring* pSubnet)
```

Argumente:

STRING* pIpAddress

Zeiger auf die Ip-Adresse

STRING* pSubnet

Zeiger auf die Subnet-Maske

Rückgabe:

BrbCallerStates_ENUM

Status als Enumeration:

BrbIpAdrType_ENUM		Ip-Adress-Typen
eBRB_IP_ADR_TYPE_UNKNOWN		Unbekannte oder ungültige Adresse
eBRB_IP_ADR_TYPE_NETWORK		Netzwerk-Adresse
eBRB_IP_ADR_TYPE_HOST		Teilnehmer-Adresse
eBRB_IP_ADR_TYPE_BROADCAST		Broadcast-Adresse

Beschreibung:

Diese Funktion gibt den Typen einer textuellen Ip-Adresse zurück. So kann z.B. die Eingabe eines Operators geprüft werden.

Hinweis: Es empfiehlt sich, die Ip-Adresse und das Subnet nach Eingabe, aber vor dem Aufruf dieser Funktion durch die Funktion `BrbCheckIpAddress` (siehe oben) berichtigen zu lassen.

Hintergrund:

Wie oben erklärt, definiert die Subnet-Maske den Bereich und damit die Anzahl der Teilnehmer (Hosts), welche innerhalb eines Netzwerks adressiert werden können. Laut internationaler Festlegung sind aber zwei Teilnehmer-Adressen dieses Bereichs für besondere Zwecke reserviert und dürfen somit keinem Teilnehmer zugewiesen werden:

-Kleinste Adresse = Netzwerk-Basis-Adresse

-Größte Adresse = Broadcast-Adresse

Die Netzwerk-Basis-Adresse dient dazu, das Netzwerk eindeutig zu identifizieren.

Die Broadcast-Adresse dient dazu, ein Telegramm an alle Teilnehmer zu senden, die sich in diesem Netzwerk befinden.

Um zu verhindern, dass eine Ethernet-Schnittstelle eine dieser reservierten Adressen innerhalb ihres Netzwerkes bekommt, kann vor der Zuweisung der Typ mit dieser Funktion ermittelt werden.

4.13.5 BrbCheckIpEqualSubnet

```
plcbit BrbCheckIpEqualSubnet(plcstring* pIp1, plcstring* pSubnet1, plcstring* pIp2, plcstring* pSubnet2)
```

Argumente:

STRING* pIp1
Zeiger auf die erste Ip-Adresse

STRING* pSubnet1
Zeiger auf das Subnet der ersten Adresse

STRING* pIp2
Zeiger auf die zweite Ip-Adresse

STRING* pSubnet2
Zeiger auf das Subnet der zweiten Adresse

Rückgabe:

BOOL

0= Die Adressen liegen nicht im selben Subnet
1= Die Adressen liegen im selben Subnet

Beschreibung:

Diese Funktion prüft zwei Ip-Adressen mithilfe deren Subnet-Angabe, ob sie sich im selben Subnet befinden. Da sich auf einem Zielsystem jede Ethernet-Schnittstelle in einem unterschiedlichen Subnet befinden muss, ist diese Funktion sehr nützlich, wenn die Ip-Adressen vom Benutzer eingegeben werden können. Es kann damit also eine ungültige Konfiguration schon vor dem Setzen der Ip-Adressen vermieden werden.

Hinweis: Es empfiehlt sich, die Ip-Adressen und die Subnets nach Eingabe, aber vor dem Aufruf dieser Funktion durch die Funktion `BrbCheckIpAddress` (siehe oben) berichtigen zu lassen.

4.14 Additional

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

4.14.1 BrbDebounceInput

```
void BrbDebounceInput(struct BrbDebounceInput* inst)
```

Argumente:

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

Eingänge:

```
BOOL bInput  
    Der zu entprellende Eingang  
UDINT nDebounceTime  
    Die Entprell-Zeit in [ms]
```

Ausgänge:

```
BOOL bOutput  
    Der entprellte Eingang
```

Beschreibung:

Entprellt einen Eingang.

Wird eine steigende Flanke am Eingang erkannt, wird der Ausgang für 1 Zyklus auf „1“ gesetzt und die Entprell-Zeit gestartet.

Eine erneute steigende Flanke am Eingang wird erst wieder berücksichtigt, wenn die Entprell-Zeit abgelaufen ist.

4.14.2 BrbGetStructureMemberOffset

```
unsigned short BrbGetStructMemberOffset(plcstring* pStructName, plcstring* pMemberName, unsigned  
long* pOffset)
```

Argumente:

```
STRING* pStructName  
    Zeiger auf den Instanz-Namen der Struktur  
STRING* pMemberName  
    Zeiger auf den Item-Name  
UDINT* pOffset  
    Zeiger auf den UDINT, der den ermittelten Offset aufnimmt
```

Rückgabe:

```
UINT  
    Rückgabe-Wert der intern aufgerufenen System-Funktionen
```

Beschreibung:

Ermittelt den Byte-Offset eines Elements innerhalb einer Struktur. Er wird benötigt, wenn auf das Element per Adresse zugegriffen werden soll.

Dazu wird eine Instanz der Struktur benötigt. Wenn es eine lokale Instanz ist, muss beim Argument „pStructName“ am Anfang der Taskname gefolgt von einem Doppelpunkt eingefügt werden („Taskname:Variablenname“).

Jeder Standard-Datentyp hat eine bestimmte Länge:

BOOL	1 Byte
USINT, SINT	1 Byte
UINT, INT	2 Bytes
UDINT, DINT, REAL	4 Bytes

Der Offset eines Items innerhalb einer Struktur kann nicht einfach durch Addieren der Längen ermittelt werden, da vom Compiler aus Effizienzgründen an den verschiedensten Stellen Füll-Bytes (Stichwort „Alignment“) eingefügt werden.

4.14.3 BrbGetCompilerVersion

```
unsigned short BrbGetCompilerVersion(struct BrbCompilerVersion_TYP* pCompilerVersion)
```

Argumente:

```
struct BrbCompilerVersion_TYP* pCompilerVersion  
Zeiger auf eine Instanz von "BrbCompilerVersion_TYP"
```

Rückgabe:

DINT

Bitcodiert:

Bit#0 = 1 = Major-Versionsnummer konnte ermittelt werden

Bit#1 = 2 = Minor-Versionsnummer konnte ermittelt werden

Bit#2 = 4 = Patch-Versionsnummer konnte ermittelt werden

0 = Keine Angaben vom System verfügbar

7 = Alle Versionsnummern konnten ermittelt werden

Beschreibung:

Gibt die Version des verwendeten GCC-Compilers in verschiedenen Formaten zurück:

BrbCompilerVersion_TYP		
nMajor	INT	Haupt-Versionsnummer
nMinor	INT	Unter-Versionsnummer
nPatch	INT	Patch-Versionsnummer
nTotal	DINT	Komplette Versionsnummer
sText	STRING[10]	Versionsnummer als Text

Bei einer Compiler-Version von 6.3.0 werden folgende Ausgaben gemacht:

CompilerVersion		
bValid	BOOL	TRUE
nMajor	INT	6
nMinor	INT	3
nPatch	INT	0
nTotal	DINT	60300
sText	STRING[10]	'6.3.0'

Die Angabe „nTotal“ sieht für jedes Segment 2 Stellen vor („0“ wird also zu „00“, „3“ wird zu „03“). Mit dieser Angabe kann dann leicht durch <= oder >= unterschieden werden.