Documentation of the library

WagoAppMem

Release 1.6.2.0
## Contents

1 Description

2 10 Documentation
   2.1 doc10_General (FB) ..................................................... 3

3 20 Program Organization Units
   3.1 Shared Memory .......................................................... 5
      3.1.1 FbSharedMemory_Base (FB) ...................................... 5
         FbSharedMemory_Base.ActualSize (PROP) ......................... 6
         FbSharedMemory_Base.Close (METH) ............................... 6
         FbSharedMemory_Base.GetDataPointer (METH) .................. 7
         FbSharedMemory_Base.IsOpen (PROP) ............................. 7
         FbSharedMemory_Base.OpenByName (METH) ....................... 8
      3.1.2 FbSharedMemory_WithReadWrite (FB) .......................... 9
         FbSharedMemory_WithReadWrite.Read (METH) ..................... 9
         FbSharedMemory_WithReadWrite.ReadByte (METH) ............. 10
         FbSharedMemory_WithReadWrite.Write (METH) ................. 11
         FbSharedMemory_WithReadWrite.WriteByte (METH) .......... 11

4 LibraryResult (GVL) .................................................... 13

5 ResultItems (GVL) ...................................................... 14

6 VersionHistory (GVL) .................................................. 15

7 Library Reference .................................................... 16
This document is automatically generated. Because of this, the chapter 30 Visualization is not shown in this document. If you are interested in getting to know more about visualization, we refer to the library manager of e!Cockpit.

Subject to Changes

WAGO Kontakttechnik GmbH & Co. KG reserves the right to provide for any alterations or modifications. WAGO Kontakttechnik GmbH & Co. KG owns all rights arising from the granting of patents or from the legal protection of utility patents. Third-party products are always mentioned without any reference to patent rights. Thus, the existence of such rights cannot be excluded.

Personnel Qualification

All tasks that are carried out with libraries made for the e!COCKPIT software must only be performed by qualified electrical specialists instructed in PLC programming according to IEC 61131-3.

All tasks that have an effect on the properties or the behavior of automation hardware or software products must only be performed by qualified employees with a thorough knowledge of handling the products concerned.

Intended Use of e!COCKPIT Libraries

Libraries created for the e!COCKPIT software are used to simplify the development of application projects in the IEC 61131-3 programming languages.

For automation tasks, WAGO offers programmable logic controllers in a wide variety of performance classes. In combination with a wide range of I/O modules, the controllers can process standard types of field signals. Controllers can be implemented centrally or in decentralized configurations. The controllers offer interfaces for the most commonly used fieldbuses for use in decentralized configurations. Fieldbus independent I/O modules are then linked via fieldbus couplers. WAGO controllers offer a runtime environment for user programs called e!RUNTIME. Software projects for implementation in e!RUNTIME environments can be created in e!COCKPIT. The programming environment in e!COCKPIT is based on the established CODESYS 3 industrial standard. Users with a previous knowledge of CODESYS 3 will thus find this environment largely familiar. The following programming languages of the IEC 61131-3 standard are available:

- Structured Text (ST)
- Ladder Diagram (LD)
- Function Block Diagram (FBD)
- Instruction List (IL)
- Sequential Function Chart (SFC)
- Continuous Function Chart (CFC)

The individual programming languages can also be combined as required during the development of the software. A portfolio of prepared libraries can be accessed for many frequently used functions in order to make software development more efficient. This document provides an overview of the WagoAppMem that WAGO offers for e!COCKPIT.
Handling application memory and shared memory

Further library information are summarized here:

- **Company**: WAGO
- **Title**: WagoAppMem
- **Version**: 1.6.2.0
- **Categories**: WAGO LayerView|App; WAGO FunctionalView|Base; Application
- **Namespace**: WagoAppMem
- **Author**: WAGO / u013972
- **Placeholder**: WagoAppMem
2.1 doc10_General (FB)

This library provides functions and FBs for low-level manipulation of raw memory and for handling shared memory. This is divided into the following main groups:

1. Classical memory manipulation (MemCopy, MemSet)
2. Packing and unpacking, i.e. stuffing bytes into long words etc.
3. Handling shared memory (‘SHM’).

Note: Although it might seem to fit into this context, handling of dynamically allocated memory (e.g. malloc(), free()) is explicitly not a subject to this library.

The first two items are represented in the sub-library ‘WagoSysPlainMem.library’. This library is formally another as this one, but it is automatically included together with ‘WagoAppMem’, so you will not have to bother with extra including.

These functions are also described in this document for the sake of convenience.

Standard Memory Operations

Standard memory operations will be handled technically via a sublibrary named WagoSysPlainMem.library. Documentation will be found in the first instance in that library, but it is also literally included in this library for the sake of convenience. That sublibrary need not be included separately because it will be included with this library automatically.

Shared Memory

This library provides support for safe usage of shared memory, i.e. creating shared memory and mapping data structures upon the raw shared memory.

There are two different ways of accessing shared memory supported by this library:

1. Procedural addressing via byte offsets:

   ```
   FB_SHM.read(udiOffset, ADR(bufferVariable), SizeOf(bufferVariable));
   FB_SHM.write(udiOffset, ADR(sourceVariable), SizeOf(sourceVariable));
   ```

   While the advantage of this paradigm consists in a generically controllable procedural access, there are obvious drawbacks for complex data structures. Namely the necessity to keep track of variable offsets and secondly the necessity to provide and update buffer variables in the application.

   This paradigm is represented by the function block `FbSharedMemory_WithReadWrite (FB)`.

2. Addressing via structured types. This paradigm starts with a structure definition of the shared memory:
type typMySharedStructure : STRUCT // just an example without use in reality
  iMyInt : INT;
  xMyFlag : BOOL;
  MyData : ARRAY [0 .. 19] OF typOtherStructures;
END_STRUCT
END_TYPE

Access to the data is done via a typed pointer property (called pData in this example):

\[
\text{If } (\text{FB\_MySHM}\text{.pData}\^{}.xMyFlag) \text{ THEN } [...] \text{ END\_IF;}
\]
\[
i := \text{FB\_MySHM}\text{.pData}\^{}.iMyInt +42;
\]
\[
\text{FB\_MySHM}\text{.pData}\^{}.MyData[6].sHello := \text{Concat}('Hallo',\text{sWorld});
\]

Advantages of this paradigm are at first the direct access to the shared memory without the need of a buffer variable and at second the automatic book-keeping of the component offsets due to the structurization of the memory block via the type definition.

This paradigm requires to derive specific child FBs from the base shared memory FBs in order to provide application specific typed data pointers. (FbSharedMemory_Base (FB), FbSharedMemory_WithReadWrite (FB)) Consequently, the shared memory objects in this library are both designed to serve as base classes for inheritance rather than for direct instantiation.

In contrast to the lower layer SysShm-Interface, the use of fixed addresses for the creation of shared-memory-objects is not supported here. The use of such objects which are created by other components, however, is supported without any restrictions.
CHAPTER 3

20 Program Organization Units

3.1 Shared Memory

3.1.1 FbSharedMemory_Base (FB)

Function

Provides base access to named shared memory objects.

Graphical Illustration

```
FUNCTION_BLOCK FbSharedMemory_Base
```

Function Description

Shared memory objects (‘SHM’) are used for inter process communication (IPC) and can be treated like non-persistent files with non-sequential access.

For usage, the user is supposed to derive his own shared memory object from this base class, e.g. as shown below. The parent FB provides the general means for accessing the data, while the inherited FB just adds code for type-safety of the access. (See the example below and for a broader introduction also the General section.)

Note (1): There is no use in instantiating this base class directly, because all means for data access are protected. Create derived FBs instead.

Note (2): There is a predefined FB FbSharedMemory_WithReadWrite derived form this base which does provide generic access to its data. Due to its genericity, data access has to take place via byte-positions instead of type definitions, so this is only intended either as fallback for simple data structures, such as single bytes or strings, or as base FB itself for deriving more specific FBs.

In a typical derivation, shared memory is mapped to existing data structures:

```
TYPE typMySharedStructure : STRUCT // just an example without use in reality
  bMyByte : BYTE;
  MyData : ARRAY [0 .. 19] OF typOtherStructures;
END_STRUCT
END_TYPE

// For accessing typified data, we extend the shared memory base FB
```
FUNCTION_BLOCK FbMySharedMemory : EXTENDS WagoSysMem.FbSharedMemory_Base;
  PROPERTY pData : POINTER TO typMySharedStructure
    pData := GetDataPointer(); // pData is given a **type**, while DataPointer is not.
END_FUNCTION_BLOCK

VAR
  MySHM : FbMySharedMemory
END_VAR

IF MySHM.OpenByname('Name_Of_Shm',SIZE(typMySharedStructure), FALSE, FALSE) THEN
  DoErrorHandling();
  EXIT;
END_IF;

MyShm^.bMyByte := 42;
InterestingNumber := MyShm^.Mydata[i].ImportantComponent;

FbSharedMemory_Base.ActualSize (PROP)

Function
Tells the actual size of shared memory (0 if not open).

Graphical Illustration

FbSharedMemory_Base.Close (METH)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Close</td>
<td>eResultCode</td>
</tr>
</tbody>
</table>

Function
Closes the shared memory object.

Graphical Illustration

3.1. Shared Memory
result codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>ENOSYS</td>
<td>This functionality is not supported by the target hardware.</td>
</tr>
<tr>
<td>EBADF</td>
<td>The object is not open.</td>
</tr>
</tbody>
</table>

**FbSharedMemory_Base.GetDataPointer (METH)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>GetDataPointer</td>
<td>POINTER TO BYTE</td>
</tr>
</tbody>
</table>

**Function**

Returns the pointer to the shared memory resource.

**Graphical Illustration**

![Graphical Illustration](image)

**Function Description**

This function returns a null-pointer if the FB is not open. The pointer is mapped into the address space of the calling application and can be used directly for pointing to structures.

**Note:** As this FB is designed as inheritance base, this method is marked PROTECTED, because under normal circumstances the user is not supposed to access the SHM directly. Instead, for user access, a derived FB is supposed to be defined which provides a proper user interface for the specific data access and uses the pointer only internally. (See general description)

**FbSharedMemory_Base.IsOpen (PROP)**

**Function**

Tells if the shared memory FB is ready for use.

**Graphical Illustration**

![Graphical Illustration](image)

**Function Description**

TRUE = ready, FALSE = not ready.
FbSharedMemory_Base.OpenByName (METH)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>OpenByName</td>
<td>eResultCode</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>sName</td>
<td>FileName</td>
<td>The global name of the resource</td>
</tr>
<tr>
<td></td>
<td>udiSize</td>
<td>UDINT</td>
<td>Minimum size of the Object</td>
</tr>
<tr>
<td></td>
<td>xRequiresSystemMemory</td>
<td>BOOL</td>
<td>Indicates interaction with non-IEC processes.</td>
</tr>
<tr>
<td></td>
<td>xAssumeExisting</td>
<td>BOOL</td>
<td>Assume that the SHM is created elsewhere.</td>
</tr>
</tbody>
</table>

Function

Opens the Sharedmemory (‘SHM’) FB for reading and writing.

Graphical Illustration

```
Method
FbSharedMemory_Base.OpenByName

sName     FileName   eResultCode
udiSize    UDINT     
xRequiresSystemMemory  BOOL     
xAssumeExisting        BOOL     
```

Function Description

**sName:**
System-wide unique name of the SHM resource. (The type definition is given in WagoTypesCommon, essentially this is a sized string.) If a resource with this name does not exist, it will be created unless the flag xAssumeExisting forbids this. Any strings which are valid as filenames would also be valid as memory identifiers. An empty string will result in EINVAL.

**udiSize:**
Required size of the resource. If the requested size cannot be granted, ENOMEM is returned. That may happen either if the resource has to be newly allocated and too little memory is available, or if the resource exists with a smaller size and it cannot be upsized. If a size of 0 or a size of 2 Gigabyte or more is requested, EINVAL is returned.

**xRequiresSystemMemory:**
TRUE indicates that the memory is to be shared not only with other IEC-Applications, but also with other independent non-IEC-processes on the PLC. (Visibility to other non IEC-Processes is mandatory.)

If no interaction with non-IEC-processes is intended, FALSE should be passed here. If you do not know that to enter here, FALSE is almost certainly the best choice.

If this property is requested but cannot be granted, the open() call returns ENOSYS when called with this flag set.

**Note (1):** If this flag is not set, the firmware may chose freely which method for providing shared memory is actually used. I.e. the shared memory is not explicitly protected from other processes, so other processes may or may not have access to this shared memory which was intended for IEC-only sharing. When ported to other hardware or firmware, however, the behaviour might silently switch to strict IEC-only shared memory without any explicit notice.

**Note (2):** When this flag is set, the call may fail with ENOSYS when the actual target does not support sharing memory between IEC and other different processes.

**xAssumeExisting:**
If this is set, the FB assumes that the SHM is created by another process. It will not implicitly create the SHM object on opening. Instead, ENOENT is returned if the SHM does not exist at all and EUNATCH is returned if the SHM exists, but is found to be attached in an incompatible way. This flag is used if an SHM-structure has to be initialized uniquely by one certain creating application. Before that is done any other application is not supposed to use the SHM. Therefore other applications must be prevented from creating the SHM via this flag.

<table>
<thead>
<tr>
<th>result codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>ENOSYS</td>
<td>This functionality is not supported by the target hardware.</td>
</tr>
<tr>
<td>ENOENT</td>
<td>The shared memory object does not exist and cannot be created.</td>
</tr>
<tr>
<td>EUNATCH</td>
<td>No memory is attached to the SHM device but a fully initialized SHM was expected.</td>
</tr>
<tr>
<td>ENOMEM</td>
<td>The requested amount of memory cannot be granted.</td>
</tr>
<tr>
<td>EINVAL</td>
<td>Invalid Parameter (e.g. name=&quot; or size=0)</td>
</tr>
</tbody>
</table>

### 3.1.2 FbSharedMemory_WithReadWrite (FB)

**Function**

Provides rudimentary read-write access to shared memory objects.

**Graphical Illustration**

![Function Block](image)

**Function Description**

This function block provides access to named shared memory objects. This FB is an extension to the base FbSharedMemory function block. It provides read and write access with explicite memory boundary checking.

Functions for opening and closing and getting properties are inherited from FbSharedMemory. New methods are:

- Read()
- ReadByte()
- Write()
- WriteByte()

**Note:** This FB provides just rudimentary methods and protection for data access in simple data structures. For serious structures, it is strongly recommended to derive more specialized FBs in order to obtain type-safety and more convenience.

(For a conceptional example see the description of the parent class FbSharedMemory_Base.)

**FbSharedMemory_WithReadWrite.Read (METH)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Read</td>
<td>eResultCode</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>udiPosition</td>
<td>UDINT</td>
<td>Offset from the zero position in SHM.</td>
</tr>
<tr>
<td></td>
<td>pRxBuffer</td>
<td>POINTER TO BYTE</td>
<td>Where to put the data</td>
</tr>
<tr>
<td></td>
<td>udiRxBufferSize</td>
<td>UDINT</td>
<td>How many bytes of data</td>
</tr>
</tbody>
</table>
Function

Reads a block of bytes from the shared memory (‘SHM’).

Graphical Illustration

Method

FbSharedMemory_WithReadWrite.Read

<table>
<thead>
<tr>
<th>udiPosition</th>
<th>UDINT</th>
<th>eResultCode</th>
<th>Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>pRxBuffer</td>
<td>Pointer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>udiRxBufferSize</td>
<td>UDINT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Function Description

A portion of the data is read from the SHM starting from the position indicated by udiPosition. It is stored locally into the address indicated by pRxBuffer (without any offsets).

udiRxBufferSize denotes the number of bytes which is to be transferred. Please note that this is not the size of the whole buffer, but the size of the substructure which is transferred.

result codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>EBADF</td>
<td>The FB is not open.</td>
</tr>
<tr>
<td>EINVAL</td>
<td>Invalid Parameters: destination address or size is 2 Gigabyte or more.</td>
</tr>
<tr>
<td>ENOSYS</td>
<td>This functionality is not supported by the target hardware.</td>
</tr>
<tr>
<td>EFAULT</td>
<td>Adressed memory is out of the range of the shared memory object.</td>
</tr>
<tr>
<td>EACCES</td>
<td>Other errors</td>
</tr>
</tbody>
</table>

FbSharedMemory_WithReadWrite.ReadByte (METH)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>ReadByte</td>
<td>BYTE</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>udiPosition</td>
<td>UDINT</td>
<td>Offset from the zero position</td>
</tr>
</tbody>
</table>

Function

Reads a single byte from shared memory (‘SHM’).

Graphical Illustration

Method

FbSharedMemory_WithReadWrite.ReadByte

<table>
<thead>
<tr>
<th>udiPosition</th>
<th>UDINT</th>
<th>BYTE</th>
<th>ReadByte</th>
</tr>
</thead>
</table>

Function Description

A data byte is read from the SHM from the position indicated by udiPosition.

If an error occurs then simply 0 is returned.

Note: As a consequence of the intentionally shortened interface, no result code is returned here. If full access to proper error handling is required, then use

```plaintext
myResult := Read(udiPosition, pData, 1);
```

instead.
FbSharedMemory_WithReadWrite.Write (METH)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Write</td>
<td>eResultCode</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>udiPosition</td>
<td>UDINT</td>
<td>Offset from the zero position in SHM</td>
</tr>
<tr>
<td></td>
<td>pTxBuffer</td>
<td>POINTER TO BYTE</td>
<td>Where to get the data from</td>
</tr>
<tr>
<td></td>
<td>udiTxNBytes</td>
<td>UDINT</td>
<td>How many bytes of data</td>
</tr>
</tbody>
</table>

Function

 Writes into a shared memory ('SHM') block.

Graphical Illustration

Method

```
FbSharedMemory_WithReadWrite.Write
  udiPosition UDINT  eResultCode  Write
  pTxBuffer  Pointer
  udiTxNBytes UDINT
```

Function Description

A number of `udiTxNBytes` data bytes are taken from the location to which `pTxBuffer` points into the local memory and is written to the position in the Shared-Memory-Object, which is indicated by `udiPosition`.

Please note that `pTxBuffer` points directly to the first byte which is transferred.

result codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>EBADF</td>
<td>The FB is not open.</td>
</tr>
<tr>
<td>EINVAL</td>
<td>Invalid parameter</td>
</tr>
<tr>
<td>ENOSYS</td>
<td>This functionality is not supported by the target hardware.</td>
</tr>
<tr>
<td>EFAULT</td>
<td>Adressed memory is out of the range of the shared memory object.</td>
</tr>
<tr>
<td>EACCES</td>
<td>Other errors</td>
</tr>
</tbody>
</table>

FbSharedMemory_WithReadWrite.WriteByte (METH)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>WriteByte</td>
<td>eResultCode</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>udiPosition</td>
<td>UDINT</td>
<td>Offset from the zero position</td>
</tr>
<tr>
<td></td>
<td>bData</td>
<td>BYTE</td>
<td>The data byte to write</td>
</tr>
</tbody>
</table>

Function

 Writes a single byte to shared memory ('SHM').

Graphical Illustration

Method

```
FbSharedMemory_WithReadWrite.WriteByte
  udiPosition UDINT  eResultCode  WriteByte
  bData       BYTE
```

3.1. Shared Memory
<table>
<thead>
<tr>
<th>result codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>EBADF</td>
<td>The FB is not open.</td>
</tr>
<tr>
<td>ENOSYS</td>
<td>This functionality is not supported by the target hardware.</td>
</tr>
<tr>
<td>EFAULT</td>
<td>Addressed memory is out of the range of the shared memory object.</td>
</tr>
<tr>
<td>EACCES</td>
<td>Other errors</td>
</tr>
</tbody>
</table>
## LibraryResult (GVL)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factory</td>
<td>FbResultFactory</td>
<td>Produces FbResults from given eResultCodes.</td>
</tr>
</tbody>
</table>

Factory for standard result objects

Use this to translate result codes from this library into standard result objects.

Usage:

```pascal
VAR
eMyResult : eResultCode;  // result code which is to be investigated
oError    : FbResult;     // result object for use in higher levels
END_VAR;

eMyResult := myFunction(...);
Namespace.LibraryResult.Factory.SetResult(eMyResult, oError);
```

(In this example ‘Namespace’ denotes the namespace which is used for including the specific library and ‘my-Function()’ is an example for a general function from this library.)
## ResultItems (GVL)

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Initial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>ERROR</td>
<td>ARRAY [0..13] OF typResultItem</td>
<td>[STRUCT(ID := OK, Severity := eSeverity.none, Text := 'OK'), STRUCT(ID := ENOSYS, Severity := eSeverity.error, Text := 'This functionality is not supported by the target hardware.'), STRUCT(ID := ENOENT, Severity := eSeverity.error, Text := 'The shared memory object does not exist and cannot be created.'), STRUCT(ID := EUNATCH, Severity := eSeverity.error, Text := 'No memory is attached to the SHM device - but was expected so.'), STRUCT(ID := ENOMEM, Severity := eSeverity.error, Text := 'The requested amount of memory cannot be granted.'), STRUCT(ID := EINVAL, Severity := eSeverity.error, Text := 'Invalid parameter (e.g. name=&quot;&quot; or size=0 or others).'), STRUCT(ID := EBADF, Severity := eSeverity.error, Text := 'FB not opened or initialized.'), STRUCT(ID :=EFAULT, Severity := eSeverity.error, Text := 'Adressed memory is not in the valid address range.'), STRUCT(ID := EACCES, Severity := eSeverity.error, Text := 'Permission denied for this resource or other errors.'), STRUCT(ID := EPERM, Severity := eSeverity.error, Text := 'This operation is not permitted in this situation.'), STRUCT(ID := EAGAIN, Severity := eSeverity.error, Text := 'Try again (a subsequent retry of the call might be successful.)'), STRUCT(ID := EBUSY, Severity := eSeverity.error, Text := 'Device or resource is busy and cannot respond to the desired function.'), STRUCT(ID := EEXIST, Severity := eSeverity.error, Text := 'Resource exists but is expected to be non-existent.'), STRUCT(ID := EALREADY, Severity := eSeverity.error, Text := 'The operation is already in progress.')]</td>
</tr>
</tbody>
</table>

**Standard result items specific for this library**

Note: This is a general mapping of result codes to short standard texts which are appropriate to the usage of these codes in this library.

Typically, each unit (function, method, or function block) in this library uses only a subset of these codes. Please, refer to the documentation of the specific unit for the set of codes which is actually used and for a detailed explanation of the meaning of a result code in the specific context.
### VersionHistory (GVL)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>WagoAppMem</td>
<td>ProjectInfo</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Author</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>08.01.2019</td>
<td>1.6.2.0</td>
<td>u015842</td>
<td>Properties: free placeholder added</td>
</tr>
<tr>
<td>18.04.2018</td>
<td>1.6.1.1</td>
<td>WAGO / u013972</td>
<td>Resolve documentation errors</td>
</tr>
<tr>
<td>10.03.2016</td>
<td>1.6.1.0</td>
<td>WAGO / u013972</td>
<td>Publish WagoSysErrorBase</td>
</tr>
<tr>
<td>04.03.2016</td>
<td>1.6.0.0</td>
<td>WAGO / u013972</td>
<td>Replace WagoAppErrorBase with WagoSysErrorBase</td>
</tr>
<tr>
<td>29.09.2015</td>
<td>1.5.2.0</td>
<td>WAGO / u013972</td>
<td>Resolve libraries with placeholder</td>
</tr>
<tr>
<td>23.09.2015</td>
<td>1.5.1.0</td>
<td>WAGO / u013972</td>
<td>Workaround for C0351-Bug</td>
</tr>
<tr>
<td>23.06.2015</td>
<td>1.5.0.0</td>
<td>WAGO / u013972</td>
<td>Release version</td>
</tr>
</tbody>
</table>
Library Reference

This is a dictionary of all referenced libraries and their name spaces.

**WagoSysErrorBase**

*Library Identification:*
- Placeholder: WagoSysErrorBase
- Default Resolution: WagoSysErrorBase, * (WAGO)
- Namespace: WagoSysErrorBase

*Library Properties:*
- LinkAllContent: False
- QualifiedOnly: False
- SystemLibrary: False
- Optional: False

*Library Parameter:*
- Parameter: RES_LOG_MAX_FILESIZE = 2000
- Parameter: RES_LOG_MAX_FILES = 1
- Parameter: RES_LOG_MAX_ENTRIES = 200
- Parameter: RES_LOG_NAME = ‘WagoAppResultLogger’

**WagoSysPlainMem**

*Library Identification:*
- Placeholder: WagoSysPlainMem
- Default Resolution: WagoSysPlainMem, * (WAGO)
- Namespace: WagoSysPlainMem

*Library Properties:*
- LinkAllContent: False
- Optional: False
- QualifiedOnly: False
- SystemLibrary: False
- PublishSymbolsInContainer: True
WagoSysShm_Internal_PFC

Library Identification:
Placeholder: WagoSysShmInternal
Default Resolution: WagoSysShm_Internal_PFC, * (WAGO)
Namespace: WagoSysShm_Internal

Library Properties:

• LinkAllContent: False
• QualifiedOnly: False
• SystemLibrary: False
• Optional: False

WagoSysStandard

Library Identification:
Placeholder: WagoSysStandard
Default Resolution: WagoSysStandard, * (WAGO)
Namespace: WagoSysStandard

Library Properties:

• LinkAllContent: False
• Optional: False
• QualifiedOnly: False
• SystemLibrary: False
• PublishSymbolsInContainer: True

WagoSysVersion

Library Identification:
Name: WagoSysVersion
Version: 1.0.0.0
Company: WAGO
Namespace: WagoSysVersion

Library Properties:

• LinkAllContent: False
• QualifiedOnly: False
• SystemLibrary: False
• Optional: False

WagoTypesCommon

Library Identification:
Placeholder: WagoTypesCommon
Default Resolution: WagoTypesCommon, * (WAGO)
Namespace: WagoTypes
Library Properties:

- LinkAllContent: False
- QualifiedOnly: False
- SystemLibrary: False
- Optional: False

WagoTypesErrorBase

Library Identification:
Placeholder: WagoTypesErrorBase
Default Resolution: WagoTypesErrorBase, * (WAGO)
Namespace: WagoTypesErrorBase

Library Properties:

- LinkAllContent: False
- Optional: False
- QualifiedOnly: True
- SystemLibrary: False
- PublishSymbolsInContainer: True

© WAGO Kontakttechnik GmbH & Co. KG, Germany 2018 – All rights reserved. For the avoidance of doubt, this copyright notice does not only apply to the information above but also and primarily to the described library itself. Please note that third-party products are always mentioned without reference to intellectual property rights, including patents, utility models, designs and trademarks, accordingly the existence of such rights cannot be excluded. WAGO is a registered trademark of WAGO Verwaltungsgesellschaft mbH.