Documentation of the library
WagoAppEvent

Release 1.6.2.0
Contents

1 Description 1

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

3 20 Program Organization Units 6
  3.1 10 Compact .................................. 6
    3.1.1 FbWagoPostEvent_cpt (FB) .................. 6
    3.1.2 FbWagoSimpleEventCatcher_cpt (FB) ........... 7
  10 Methods ...................................... 8
    FbWagoSimpleEventCatcher_cpt.protRun (METH) ........ 8
  3.2 30 Base ..................................... 9
    3.2.1 FbWagoEventCatcher (FB) .................... 9
    01 Main Interface ............................... 9
      FbWagoEventCatcher.Register (METH) ............ 9
      FbWagoEventCatcher.Unregister (METH) .......... 10
      FbWagoEventCatcher.UnregisterAll (METH) ....... 10
    02 Auxiliary .................................. 11
      FbWagoEventCatcher.FreeRegistrationSlots (PROP) 11
      FbWagoEventCatcher.GetEventCount (METH) ....... 11
    03 Protected ................................... 11
      FbWagoEventCatcher.protRun (METH) ............. 11
    04 Administration ............................... 12
      FbWagoEventCatcher.Finish (METH) ............... 12
      FbWagoEventCatcher.Initialize (METH) .......... 13
    3.2.2 WagoPostEvent (FUN) ..................... 13
  3.3 40 Auxiliary Functions ........................ 14
    3.3.1 ComposeEventID (FUN) ..................... 14
    3.3.2 DoesEventExist (FUN) ..................... 15
    3.3.3 GetNumberOfRegisteredCallbacks (FUN) .... 15

4 29 Types 16
  4.1 eWagoEventClass (ENUM) ..................... 16
  4.2 typWagoEventInfo (STRUCT) ................. 16

5 LibraryResult (GVL) 18

6 ParameterList (PARAMS) 19

7 ResultItems (GVL) 20

8 VersionHistory (GVL) 21
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 WagoAppEvent that WAGO offers for e!COCKPIT.
Handling Events

Further library information are summarized here:

- **Company**: WAGO
- **Title**: WagoAppEvent
- **Version**: 1.6.2.0
- **Categories**: WAGO LayerView\App; Application; WAGO FunctionalView\Base
- **Author**: WAGO / u013972
- **Placeholder**: WagoAppEvent
CHAPTER 2

10 Documentation

2.1 doc10_General (FB)

The library 'WagoAppEvent.library' provides functionalities for registering callbacks to system events and for triggering ('posting') application specific events.

Context

The major differences / advantages to the more fundamental 'CmpEvenrMgr.library' are the following:

• In 'WagoAppEvent', the set of usable system events is restricted to those events, which are considered 'safe' for application usage and which are also supported by the target hardware. These restrictions are represented in an internal target specific sub-library 'WagoSysEvent_Internal_<target>', which is selected by the device description.

• The callback function blocks carry all functionality for administrative information internally. The application does not have to care about handles or double registration. Such a callback function block can be registered to more than one event simultaneously. Unregistering is done automatically when the FB is unloaded.

• Event identifications and parameter descriptions are available in a re-organized form directly from this library. This re-organization is in most cases a direct 1:1-mapping of the constants which could be found in other system libraries, but it is extended by Wago-specific events and it is cleaned from those system constants which are considered to lead to unsafe usage. No other libraries than 'WagoAppEvent' itself have to be included.

• Expected return values from the callback methods are handled in a unified and also formalized way (i.e. by a pointer to the range where the return values are to be written to and a size specification). In simple and well documented cases (e.g. 'deny'-events), the 'yes/no'-decision made by the callback function is transported via the unified result code of the callback ('OK' / other than 'OK') and it is inserted into the proper places automatically by the underlying framework of this library. All this avoids mistakes which would be hard to debug.

• Parameters of the event are given in plain form to the callback functions without the necessity to parse chains of pointers.

• There are compact function blocks available which implement the standard Wago behaviour models and which are easy-to-use in graphical development environments.

Usage

For using callbacks, the application would derive its own callback function block from FbWagoEventcatcher(FB) of this library and redefine the runner method 'protRun()' to the specific function:

Redefinition of Callback-Method:
New FB handles application event STOP: (inherits from base class)

FUNCTION_BLOCK FbEventCatcher_Stop EXTENDS FbWagoEventCatcher

Common interface of Wago event callback

METHOD PROTECTED protRun : eResultCode // This is the callback method for STOP
  VAR_INPUT
    Info : typWagoEventInfo; // it returns EACCESS when it does not.
  END_VAR

Application specific content of callback method

CASE Info.eEventID OF
  // If registered to more than one event, find out which of these events has been issued.
  eWagoEventID.PrepareStop:
    Notify_The_Application(Info.unParameter.CmpAppStop.eStopReason);
    protRun := OK;
  eWagoEventID.DenyStop:
    // Decide if stopping is ok or not
    IF Get_Allowance_From_Application_To_Stop() THEN
      protRun := OK; // confirm the allowance to stop
    ELSE
      protRun := EACCESS; // deny the stop request
    END_IF
  ELSE:
    // other events. Just pass through.
    protRun := OK;
END_CASE;

Note: We use a new method name ‘protRun’ instead of the original callback name ‘EventCallback()’, because
  • EventCallback() is already used for internal administration, such as parameter handling and wrapping,
  • EventCallback() has a different (more obscure) interface
  • EventCallback() cannot be declared PROTECTED as it should be, because the callback should only be called from the event, never from the application itself.

Automatic Initialization within Catcher FB:

In the previous step we have defined what the event catcher has to do in case of an event. In the following steps, we now demonstrate how to register that catcher to the system, so that events are actually passed to it.

The easiest way would be to handle registering directly within the initialization method of the FB. In this case, no further calls are necessary within the application for registering the event. The initialize() method is called from FB_INIT() and will be executed during internal construction of the FB.

FUNCTION_BLOCK FbEventCatcher_Stop EXTENDS FbWagoEventCatcher  // handles application stop

METHOD initialize

SUPER^.initialize(); // original initialization. Be careful to insert this!
Register(eWagoEventID.DenyStop); // additional: registering an event
Register(eWagoEventID.PrepareDone); // additional: registering another event
Application Code:

Within the application code, the catcher FB has merely to be constructed:

```pascal
VAR
  FbStopHandler : FbEventCatcher_Stop;
END_VAR
```

Basically, no further registering code is necessary. Nevertheless, the catcher might also be registered from within the application. That would allow for a more detailed control about when the registration takes place:

```pascal
IF not xInitialized THEN
  FbStopHandler.Register(eWagoEventID.PrepareStop);
  FbStopHandler.Register(eWagoEventID.DenyStop);
  FbStopHandler.Register(...);
  xInitialized := TRUE;
END_IF
```

Both ways could be mixed. When registering only from the application side, it is not necessary to redefine the method initialize().
3.1 10 Compact

3.1.1 FbWagoPostEvent_cpt (FB)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
<th>Inherited from</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>oStatus</td>
<td>FbResult</td>
<td>Status object (see WagoSysErrorBase).</td>
<td>FbBehaviour-Model_oStatus_Base</td>
</tr>
<tr>
<td>Inout</td>
<td>xTrigger</td>
<td>BOOL</td>
<td>Triggers the action and signals termination.</td>
<td>FbBehaviour-Model_WagoAppTrigger</td>
</tr>
<tr>
<td>Output</td>
<td>xError</td>
<td>BOOL</td>
<td>Indicates that an error has occurred.</td>
<td>FbBehaviour-Model_WagoAppTrigger</td>
</tr>
<tr>
<td></td>
<td>xBusy</td>
<td>BOOL</td>
<td>True while not terminated.</td>
<td>FbBehaviour-Model_WagoAppTrigger</td>
</tr>
<tr>
<td>Input</td>
<td>eEventID</td>
<td>eWagoEventID</td>
<td>Identification number for the event</td>
<td></td>
</tr>
<tr>
<td></td>
<td>eParameterID</td>
<td>eWagoEventParameterID</td>
<td>Identifies the parameter structure for the event</td>
<td></td>
</tr>
<tr>
<td></td>
<td>wParamVersion</td>
<td>WORD</td>
<td>Additional versioning for parameter structure.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pParameter</td>
<td>POINTER TO unWagoEventParameter</td>
<td>The parameter structure itself</td>
<td></td>
</tr>
</tbody>
</table>

Function

This is a FUP-oriented function block for posting events.

Graphical Illustration

Function Block

Function Description

This FB implements the behaviour model ‘WagoAppTrigger’.
### Result Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK = 0</td>
<td>Event was registered successfully</td>
</tr>
<tr>
<td>EPERM</td>
<td>This event must not be posted via WagoAppEvent</td>
</tr>
<tr>
<td>ENOENT</td>
<td>The event-ID was unknown</td>
</tr>
<tr>
<td>(others)</td>
<td>Other problems while posting the event</td>
</tr>
</tbody>
</table>

**Note:** For simply posting events without parameters, the inputs eParameterID, wParamVersion, and pParameter may just be set to 0.

When parameters have to be passed, it depends on the individual eWagoEventID which values for eParameterID, wParamVersion, and unWagoEventParameter are acceptable or valid. For a detailed description, please refer to the documentation of WagoTypesEvent and to the 3S-description of these events.

### 3.1.2 FbWagoSimpleEventCatcher_cpt (FB)

#### Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
<th>Inherited from</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>oStatus</td>
<td>FbResult</td>
<td>Status object (see WagoSysErrorBase).</td>
<td>FbBehaviour-Model_oStatus_Base</td>
</tr>
<tr>
<td></td>
<td>xOpen</td>
<td>BOOL</td>
<td>Desired channel state (open=TRUE, closed=FALSE)</td>
<td>FbBehaviour-Model_WagoAppChannel</td>
</tr>
<tr>
<td>Output</td>
<td>xError</td>
<td>BOOL</td>
<td>Feedback: an Error has occurred.</td>
<td>FbBehaviour-Model_WagoAppChannel</td>
</tr>
<tr>
<td></td>
<td>xIsOpen</td>
<td>BOOL</td>
<td>Feedback: channel is operative.</td>
<td>FbBehaviour-Model_WagoAppChannel</td>
</tr>
<tr>
<td></td>
<td>xIsIdle</td>
<td>BOOL</td>
<td>Feedback: channel is clear for new opening.</td>
<td>FbBehaviour-Model_WagoAppChannel</td>
</tr>
<tr>
<td>Input</td>
<td>eEventID</td>
<td>eWagoEventID</td>
<td>the event number which is to be regarded.</td>
<td></td>
</tr>
</tbody>
</table>

#### Function

This is a FUP-oriented function block for catching single events.

#### Graphical Illustration

![Function Block Diagram](image)

#### Function Description

This function block implements the behaviour model ‘WagoAppChannel’. For using the catcher FB, you are supposed to derive your own FB from this library FB and overload the method protRun with your specific functionality.

(In the introducing section ‘General’ an example is given. Although the example is formally written for the parent this FB (i.e. FbWagoEventCatcher instead of FbWagoSimpleEventCatcher_cpt), the code which is to be written for using any of these FBs is identical, because both FB differ only in the interface variables of the behaviour model.)

The xOpen-Input controls the registering and unregistering of the event.)
When the registered event is caught, the protected method `protRun()` is executed. This runner method is provided with the full set of event parameters via its input `Info: typWagoEventInfo`.

Feedback from the runner is provided in two ways:

1. If the parameter format indicates a ‘deny’-Option, the result code of the runner determines, whether the event is responded with ‘deny’ (code<>0) or ‘accept’ (code=0).
2. The parameter structure contains a pointer to a feedback area and the size of the feedback area. If one of these is zero, no feedback is fed back.

**Note:** If multiple events are to be mapped on a single Catching FB, the method-oriented `FbWagoEventCatcher(FB)` should be used instead.

**Attention:** Take care to have `SUPER^()` involved in the body of derived FBs, as otherwise the behaviour model will not work.

### Result Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK = 0</td>
<td>Event was registered successfully.</td>
</tr>
<tr>
<td>EPERM</td>
<td>The event must not be registered via WagoAppEvent.</td>
</tr>
<tr>
<td>EKEYREJECTED</td>
<td>The event-ID was unknown and could not be created.</td>
</tr>
<tr>
<td>ENOENT</td>
<td>The event-ID was unknown and not attempt was made to create it (future use).</td>
</tr>
<tr>
<td>EACCES</td>
<td>The event could not be opened.</td>
</tr>
<tr>
<td>(others)</td>
<td>The callback could not be registered.</td>
</tr>
</tbody>
</table>

### 10 Methods

**FbWagoSimpleEventCatcher_cpt.protRun (METH)**

#### Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>protRun</td>
<td>eResultCode</td>
</tr>
<tr>
<td>Input</td>
<td>Info</td>
<td><code>typWagoEventInfo</code></td>
</tr>
</tbody>
</table>

#### Function

This is a runner method template for derived FBs of `FbWagoSimpleEventCatcher_cpt (FB)`.

#### Graphical Illustration

![Graphical Illustration of FbWagoSimpleEventCatcher_cpt.protRun]

#### Function Description

This method is called, when the event is caught. Overload it with your own code.

The input structure `Info (typWagoEventInfo)` provides information about the event identity, the parameter format and possible feedback locations.

If the event catcher is expected to make a deny/allow decision, this method is expected to signalize ‘allow’ by a return value of 0=’OK’ and ‘deny’ by any other value (e.g. EACCESS).

#### Result Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK = 0</td>
<td>Regular termination of callback runner.</td>
</tr>
<tr>
<td>(other)</td>
<td>In a ‘deny’ context: deny the requested action. Else: ignored.</td>
</tr>
</tbody>
</table>
More information about the general usage of this FB is given in the subsection ‘Usage’ of the introducing section ‘General’, which applies as well for the ‘_cpt’ FB.

3.2 30 Base

3.2.1 FbWagoEventCatcher (FB)

Function

This is a Function block base for catching events.

Graphical Illustration

![FunctionBlock](FbWagoEventCatcher)

Function Description

Usage: Derive child FBs from this base and overload the protected method `protRun()` as demonstrated in the ‘General’ overview.

When the event is posted, the protected method `protRun(typWagoEventInfo)` is executed. This runner method is provided with the full set of event parameters via its input Info: typWagoEventInfo

Feedback from the runner is provided in two ways:

1. If the parameter format indicates a “deny”-Option, the result code of the runner determines, if the event responses with “deny” (code<>0) or “accept” (code=0).

2. The parameter structure contains a pointer to a feedback area and a size of the feedback area. If one of these is zero, no feedback is fed back.

01 Main Interface

FbWagoEventCatcher.Register (METH)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Register</td>
<td>eResultCode</td>
</tr>
<tr>
<td>Input</td>
<td>eEventID</td>
<td>eWagoEventID</td>
</tr>
</tbody>
</table>

Function

Registering this FB to the specified event.

Graphical Illustration

![Method](FbWagoEventCatcher_Register)

Function Description

Note: More than one event can be registered to this FB simultaneously. The maximum number is given by ParameterList.cuiMaxEventsPerCatcher.
If an event does not exist at the time of registering, it will be created automatically. In this case it will be deleted automatically when the event is unregistered again.

<table>
<thead>
<tr>
<th>Result Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK=0</td>
<td>Event was registered successfully.</td>
</tr>
<tr>
<td>EPERM</td>
<td>The Event must not be registered via WagoAppEvent.</td>
</tr>
<tr>
<td>EKEYREJECTED</td>
<td>The Event-ID was unknown and also could not be created.</td>
</tr>
<tr>
<td>ENOENT</td>
<td>The Event-ID was unknown and no attempt was made to create it future use).</td>
</tr>
<tr>
<td>EACCES</td>
<td>The Event could not be opened.</td>
</tr>
<tr>
<td>EALREADY</td>
<td>The Event is already registered for this catcher.</td>
</tr>
<tr>
<td>ENFILE</td>
<td>Too many events have been registered on this FB.</td>
</tr>
<tr>
<td>(others)</td>
<td>The callback could not be registered due to other reasons.</td>
</tr>
</tbody>
</table>

**FbWagoEventCatcher.Unregister (METH)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Unregister</td>
<td>eResultCode</td>
</tr>
<tr>
<td>Input</td>
<td>eEventID</td>
<td>eWagoEventID</td>
</tr>
</tbody>
</table>

**Function**

Unregister one event from the FB.

**Graphical Illustration**

```
Method
FbWagoEventCatcher.Unregister
  eEventID eWagoEventID eResultCode Unregister
```

**Result Codes**

<table>
<thead>
<tr>
<th>Result Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK=0</td>
<td>Operation successfully completed</td>
</tr>
<tr>
<td>ENOENT</td>
<td>The specified event has not been registered to this FB.</td>
</tr>
<tr>
<td>EADV</td>
<td>Advertise Error: Registered Interface was evidently not registered any more.</td>
</tr>
<tr>
<td>EBADF</td>
<td>The event was not known to the system.</td>
</tr>
</tbody>
</table>

**FbWagoEventCatcher.UnregisterAll (METH)**

**Interface variables**

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

**Function**

Unregisters all events from this FB.

**Graphical Illustration**

```
Method
FbWagoEventCatcher.UnregisterAll
  eResultCode UnregisterAll
```
Result Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK=0</td>
<td>Operation successfully completed</td>
</tr>
<tr>
<td>ENOENT</td>
<td>The specified event has not been registered to this FB</td>
</tr>
<tr>
<td>EADV</td>
<td>Advertise Error: Registered Interface was evidently not registered any more</td>
</tr>
<tr>
<td>EBADF</td>
<td>Event was not known to the system</td>
</tr>
</tbody>
</table>

02 Auxiliary

FbWagoEventCatcher.FreeRegistrationSlots (PROP)

Function

The number of events which could be registered by this FB

Graphical Illustration

**Property**

![Property Diagram]

FbWagoEventCatcher.GetEventCount (METH)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>GetEventCount</td>
<td>LINT</td>
</tr>
<tr>
<td>Input</td>
<td>eEventID</td>
<td>eWagoEventID</td>
</tr>
</tbody>
</table>

Function

Returns how often an event has been caught by this FB.

Graphical Illustration

**Method**

![Method Diagram]

Function Description

The number of events of type eEventID is counted since registration. If the event has not been registered by this FB, -1 is returned.

If only one event is registered and “0” is passed for the event-ID, the count of the only registered event is returned.

03 Protected

FbWagoEventCatcher.protRun (METH)

Interface variables
Function

Runner method template for derived FBs.

Graphical Illustration

```
Method
FbWagoEventCatcher.protRun
```

Function Description

This method is called, when the event is caught. Overload it with your own code.

The input structure `Info (typWagoEventInfo)` provides information about the event identity, the parameter format and possible feedback locations.

<table>
<thead>
<tr>
<th>Component Name</th>
<th>type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>eEventID</td>
<td>eWagoEventID</td>
<td>ID of the event.</td>
</tr>
<tr>
<td>dwEventProvider</td>
<td>DWORD</td>
<td>Provider identification (CmpApp = 2)</td>
</tr>
<tr>
<td>pReturnValues</td>
<td>POINTER TO BYTE</td>
<td>Pointer to possible return values from the runner (0=none)</td>
</tr>
<tr>
<td>udiReturnSize</td>
<td>UDINT</td>
<td>Size of the feedback buffer (0=no feedback allowed)</td>
</tr>
</tbody>
</table>
| eParameterID   | eWagoEventParame-
| terID          | rterID             | Identifies the format of the event parameters                           |
| unParameter    | unWagoEventParameter| Union containing all known parameter formats                            |

If the event catcher is expected to make a deny/allow decision, this method is expected to signalize ‘allow’ by a return value of 0=‘OK’ and ‘deny’ by any other value (e.g. EACCESS).

<table>
<thead>
<tr>
<th>Result Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK = 0</td>
<td>Regular termination of callback runner.</td>
</tr>
<tr>
<td>(other)</td>
<td>In a “deny” context: deny the requested action. Else: ignored.</td>
</tr>
</tbody>
</table>

More information about the general usage of this FB is given in the subsection ‘Usage’ of the introducing section ‘General’.

04 Administration

FbWagoEventCatcher.Finish (METH)

Interface variables

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

Function

This method is called when the FB is unloaded.

Graphical Illustration
Function Description

This method is meant to be re-defined when additional cleanup code is required. Remember to call

```
SUPER^.finish()
```

in the child’s finish()-method.

**FbWagoEventCatcher.Initialize (METH)**

**Function**

This method is called when the FB is constructed.

**Graphical Illustration**

```
Method
FbWagoEventCatcher.Initialize
```

Function Description

You might redefine this method if required and insert some initializing registrations here:

```
SUPER^.initialize(); // Original initialization. Always insert this!
Register(eWagoEventID.DenyStop); // Additional: registering an event
Register(eWagoEventID.PrepareDone); // Additional: registering another event
```

Remember to call `SUPER^.initialize()` in the first line of the child’s initialize()-method.

### 3.2.2 WagoPostEvent (FUN)

**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>Wago-PostEvent</td>
<td>eResultCode</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>eEventID</td>
<td>eWagoEventID</td>
<td>The identification number of the event</td>
</tr>
<tr>
<td></td>
<td>eParameterID</td>
<td>eWagoEventParameterID</td>
<td>Format of parameter union (0=none)</td>
</tr>
<tr>
<td></td>
<td>wParamVersion</td>
<td>WORD</td>
<td>Version of parameter format (0=none)</td>
</tr>
<tr>
<td></td>
<td>pParameter</td>
<td>POINTER TO unWagoEventParameter</td>
<td>Parameters, passed to the event (0=none)</td>
</tr>
</tbody>
</table>

**Function**

Posts an event to the system.

**Graphical Illustration**
WagoAppEvent, Release 1.6.2.0

Function WagoPostEvent

<table>
<thead>
<tr>
<th>Parameter ID</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>eEventID</td>
<td>eWagoEventID</td>
<td></td>
</tr>
<tr>
<td>eParameterID</td>
<td>eWagoEventParameterID</td>
<td></td>
</tr>
<tr>
<td>wParamVersion</td>
<td>WORD</td>
<td></td>
</tr>
<tr>
<td>pParameter</td>
<td>Pointer</td>
<td></td>
</tr>
</tbody>
</table>

Function Description

Result Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK = 0</td>
<td>Event was posted successfully.</td>
</tr>
<tr>
<td>EPERM</td>
<td>The Event must not be posted via WagoAppEvent.</td>
</tr>
<tr>
<td>ENOENT</td>
<td>The Event-ID was unknown.</td>
</tr>
<tr>
<td>(others)</td>
<td>Other undocumented problems while posting the event.</td>
</tr>
</tbody>
</table>

Note: For simply posting events without parameters, the inputs eParameterID, wParamVersion, and pParameter may simply be set to 0.

When parameters have to be passed, it depends on the individual eWagoEventID which values for eParameterID, wParamVersion, and unWagoEventParameter are acceptable or valid. For more details, please refer to the Appendix ‘Types’, to the documentation of WagoTypesEvent or to the 3S-description of these events.

3.3 40 Auxiliary Functions

3.3.1 ComposeEventID (FUN)

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>ComposeEventID</td>
<td>eWagoEventID</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>EventNumber</td>
<td>WORD</td>
<td>event number</td>
</tr>
<tr>
<td></td>
<td>EventClass</td>
<td>eWagoEventClass</td>
<td>class number</td>
</tr>
</tbody>
</table>

Function

Composes a fully qualified WAGO event ID.

Graphical Illustration

Function Description

This function composes a fully qualified WAGO event ID from a 16-Bit arbitrary event number and a 16-bit Class number. It is intended to be used as a helper function when own events are created.

The Event number is meant to identify the individual number of the event, while the event class carries information about the severity of that event, e.g. ‘info’, ‘warning’, ‘exception’ or others.

When only predefined events are used this helper is not needed.
3.3.2 DoesEventExist (FUN)

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>DoesEventExist</td>
<td>BOOL</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>eEventID</td>
<td>eWagoEventID</td>
<td>The event to be investigated</td>
</tr>
</tbody>
</table>

Function

Checks if an event is known to the system or not.

Graphical Illustration

Function Description

The Function ‘DoesEventExist’ returns TRUE if the event is known to the system and FALSE otherwise.

3.3.3 GetNumberOfRegisteredCallbacks (FUN)

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>GetNumberOfRegisteredCallbacks</td>
<td>UDINT</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>eEventID</td>
<td>eWagoEventID</td>
<td>the event which is to be investigated</td>
</tr>
</tbody>
</table>

Function

This returns the number of Callbacks which are registered to a given event

Graphical Illustration
Note: Some type definitions in this library are taken from a separate sub-library ‘WagoTypesEvent.library’ which is included automatically when ‘WagoAppEvent’ is included. As these definitions are commonly used in this library they are repeated here for the sake of convenience.

### 4.1 eWagoEventClass (ENUM)

<table>
<thead>
<tr>
<th>Name</th>
<th>Initial</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>info</td>
<td>EVTCLASS_INFO</td>
<td></td>
</tr>
<tr>
<td>warning</td>
<td>EVTCLASS_WARNING</td>
<td></td>
</tr>
<tr>
<td>error</td>
<td>EVTCLASS_ERROR</td>
<td></td>
</tr>
<tr>
<td>exception</td>
<td>EVTCLASS_EXCEPTION</td>
<td></td>
</tr>
<tr>
<td>user</td>
<td>EVTCLASS_VENDOR_SPEC</td>
<td></td>
</tr>
<tr>
<td>EVTCLASS_FUSE</td>
<td>16#1001</td>
<td>Eventclass reserved for use in WagoSysFuse_Internal</td>
</tr>
<tr>
<td>all</td>
<td>EVTCLASS_ALL</td>
<td></td>
</tr>
</tbody>
</table>

Predifined values for specifying the class of a user defined event.

This type is used for composing own eventIDs with the function ‘ComposeEventID()’. It is a mere wrapper which uses a subset of the predefined codesys definitions.

### 4.2 typWagoEventInfo (STRUCT)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>eEventID</td>
<td>eWagoEventID</td>
<td>ID of the event.</td>
</tr>
<tr>
<td>dwEvent-Provider</td>
<td>DWORD</td>
<td>provider identification (CmpApp = 2)</td>
</tr>
<tr>
<td>pReturnValues</td>
<td>POINTER TO BYTE</td>
<td>pointer to possible return values from the runner (0=none)</td>
</tr>
<tr>
<td>udiReturnSize</td>
<td>UDINT</td>
<td>size of the feedback buffer (0=no feedback allowed)</td>
</tr>
<tr>
<td>eParameterID</td>
<td>eWagoEventParameterID</td>
<td>identifies the format of the event parameters</td>
</tr>
<tr>
<td>unParameter</td>
<td>unWagoEventParameter</td>
<td>union containing all known parameter formats.</td>
</tr>
</tbody>
</table>
This structure is passed as input argument to every callback runner.
The ‘eEventID’ identifies which event is caught. Although events are registered separately, multiple events may finally be handled by one single handler. These multiple events are now differentiated by this means.

When parameters are passed to the catcher, this is done via the structure ‘unParameter’, which is a union of multiple different known parameter structures. The struct-component eParameterID identifies which component of the union applies here.

When feedback is expected from a runner, there are two different ways to handle that:

1. Simple ‘accept/deny’ decisions are signaled via the return value of the runner. Feedback through the typWagoEventInfo-structure does not take place.

2. When feedback of larger size is expected, in this structure the component ‘pReturnValues’ indicates where to write that feedback and ‘udiReturnSize’ indicates the size of that area.
CHAPTER 5

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 ‘myFunction()’ is an example for a general function from this library.)
### ParameterList (PARAMS)

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Initial</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>cuiMaxEventsPer-Catcher</td>
<td>UINT</td>
<td>60</td>
<td>Determines how many events could be registered per catcher FB</td>
</tr>
<tr>
<td></td>
<td>cudiProviderID</td>
<td>UDINT</td>
<td>2</td>
<td>Pre-set identification number for the event producer</td>
</tr>
</tbody>
</table>
### 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..10] OF typResultItem</td>
<td>[STRUCT(ID := OK, Severity := eSeverity.none, Text := 'OK'), STRUCT(ID := EALREADY, Severity := eSeverity.info, Text := 'The event is already registered for this catcher.'), STRUCT(ID := EDEFAULT, Severity := eSeverity.info, Text := 'No action has been started so far.'), STRUCT(ID := EPERM, Severity := eSeverity.error, Text := 'This event must not be posted or registered via WagoAppEvent.'), STRUCT(ID := ENOENT, Severity := eSeverity.error, Text := 'The event is unknown in this context is not registered to this FB.'), STRUCT(ID := EKEYREJECTED, Severity := eSeverity.error, Text := 'The event-ID was unknown and could not be created.'), STRUCT(ID := EACCES, Severity := eSeverity.error, Text := 'The event could not be opened.'), STRUCT(ID := ENFILE, Severity := eSeverity.error, Text := 'Too many events have been registered on this FB.'), STRUCT(ID := EADV, Severity := eSeverity.error, Text := 'Advertise Error: Registered interface was evidently not registered any more.'), STRUCT(ID := EBADF, Severity := eSeverity.error, Text := 'The event was not known to the system at all.'), STRUCT(ID := EINVAL, Severity := eSeverity.error, Text := 'The combination of FbWagoPostEvent or FbWagoPostEvent_cpt with eEventID := eWagoEventID.DeleteDone and pParameter := 0 is not allowed')]</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.
CHAPTER 8

VersionHistory (GVL)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Info</th>
<th>Date</th>
<th>Version</th>
<th>Author</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>WagoAppEvent</td>
<td></td>
<td></td>
<td>08.01.2019</td>
<td>1.6.2.0</td>
<td>u015842</td>
<td>Properties: free placeholder added</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10.10.2017</td>
<td>1.6.1.1</td>
<td>WAGO / u013972</td>
<td>WagoPostEvent()/WagoPostEvent_cpt() - Workaround for WAT24385</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10.03.2016</td>
<td>1.6.1.0</td>
<td>WAGO / u013972</td>
<td>Publish WagoSysErrorBase</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>04.03.2016</td>
<td>1.6.0.0</td>
<td>WAGO / u013972</td>
<td>Replace WagoAppErrorBase with WagoSysErrorBase</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>15.02.2016</td>
<td>1.5.3.1</td>
<td>WAGO / u013972</td>
<td>Add a new EventClass</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>29.09.2015</td>
<td>1.5.2.0</td>
<td>WAGO / u013972</td>
<td>Resolve libraries with placeholder</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>23.09.2015</td>
<td>1.5.1.0</td>
<td>WAGO / u013972</td>
<td>Workaround for C0351-Bug</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>23.06.2015</td>
<td>1.5.0.0</td>
<td>WAGO / u013972</td>
<td>Release version</td>
</tr>
</tbody>
</table>

Known Bugs - WAT24385:
This is a dictionary of all referenced libraries and their name spaces.

**CmpErrors**
*Library Identification:*
Placeholder: CmpErrors
Default Resolution: CmpErrors, * (System)
Namespace: CmpErrors

*Library Properties:*

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

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

*Library Properties:*

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

**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'

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

Library Properties:
• LinkAllContent: False
• QualifiedOnly: False
• SystemLibrary: False
• Optional: False

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

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

**WagoSysTypedefs_Debugging**

*Library Identification:*
PlaceHolder: WagoSysTypedefs_Debugging
Default Resolution: WagoSysTypedefs_Debugging, * (WAGO)
Namespace: WagoSysTypedefs_Debugging

*Library Properties:*

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

**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

WagoTypesEvent

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

Library Properties:

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

© WAGO Kontaktechnik 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.