Table of Contents

Documentation of mbc engineering GmbH PCS Library

Getting Started

About

This Library will help or customer and the mbc team to interagte with the Process Control System (PCS) of mbc engineering GmbH. There are some default exchange logic and structures for reuse. This Library will help you on PLC side on some point. Example on handling PCS commands.

For code changes history looking at changelog.

Install from Library file

The library must be installed locally. The first time or on a new computer it must be installed the correct library version. For mor details about library see at infosys - Using libraries.

For Installation go to *PLC -> Library Repository* and then press the *Install* button. Choose the Library file to install. In this case choose the file `Mbc_Tc3_Pcs_v1.0.0.0.library`. TwinCat3 will it install on the default behavior to the `system` repository into the folder `C:\TwinCAT\3.1\Components\Plc\Managed Libraries\mbc engineering GmbH\MBC TC3 PCS Library\1.0.0.0`.

> More infos at infosys - Library installation

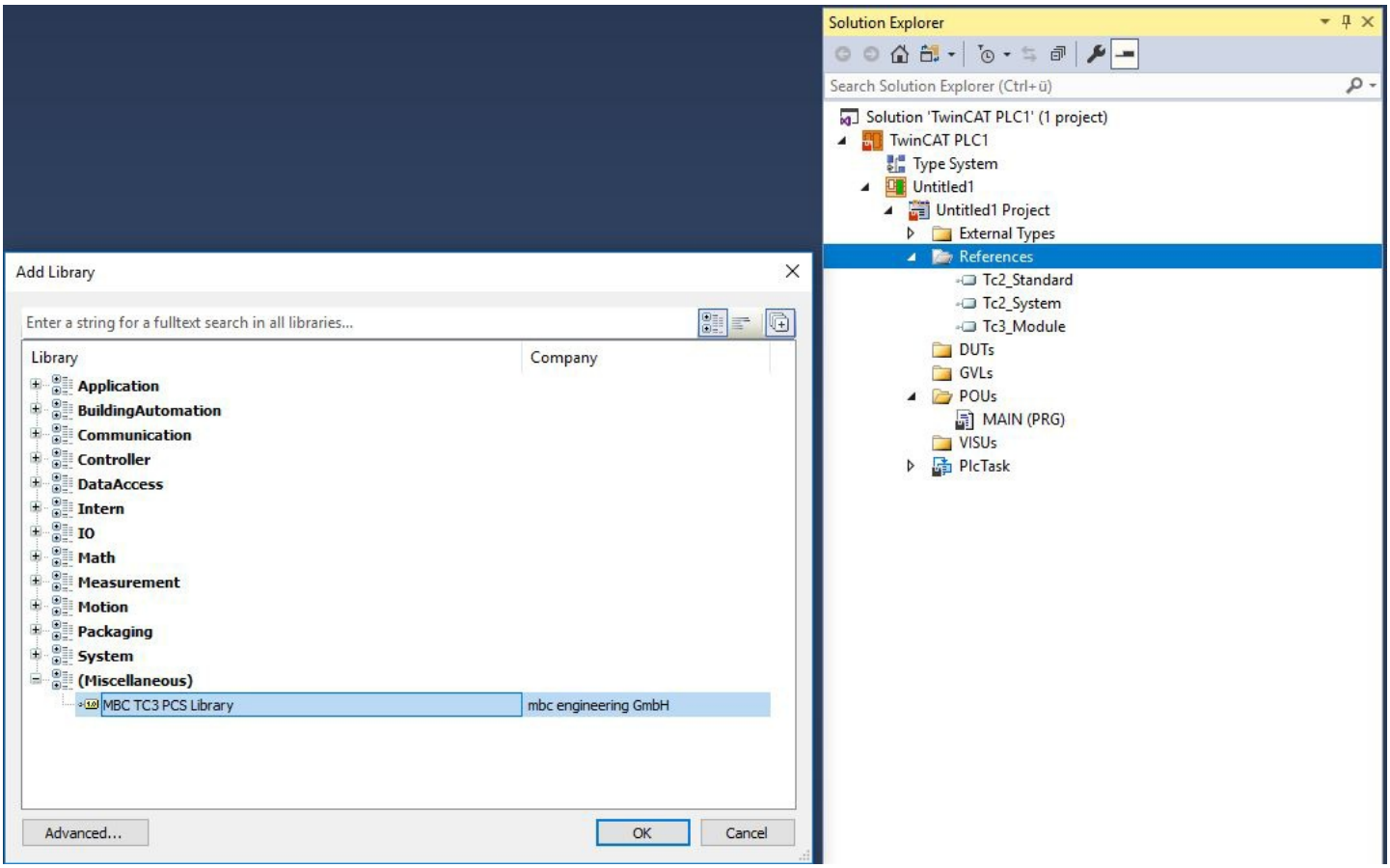
Install from a existing PLC Project

If you move a existing PLC Project that use this Library to another computer, it is possible to install it locally in TwinCat 3. This because TwinCat 3 saves all used Library in the PLC project in the project folder under `..\TwinCAT PLC1\Untitled1\_Libraries`

For local installation go to *TwinCAT PLC1 - Untitled1 -> right click -> Install Project Libraries*

Add the Library Reference to the PLC Project

When the library is installed, it is possible add the Library reference in the PLC project. go to *TwinCAT PLC1 - Untitled1 - Untitled1 Project - References -> right click -> Add Library*. In the list search for *MBC TC3 PCS Library* following press *OK*. Now the library can be used in the Project.



We recommend to use placeholder reference with the *always newest version* set.

More infos at infosys - Library Manager

In a Nutshell

- CommandBase

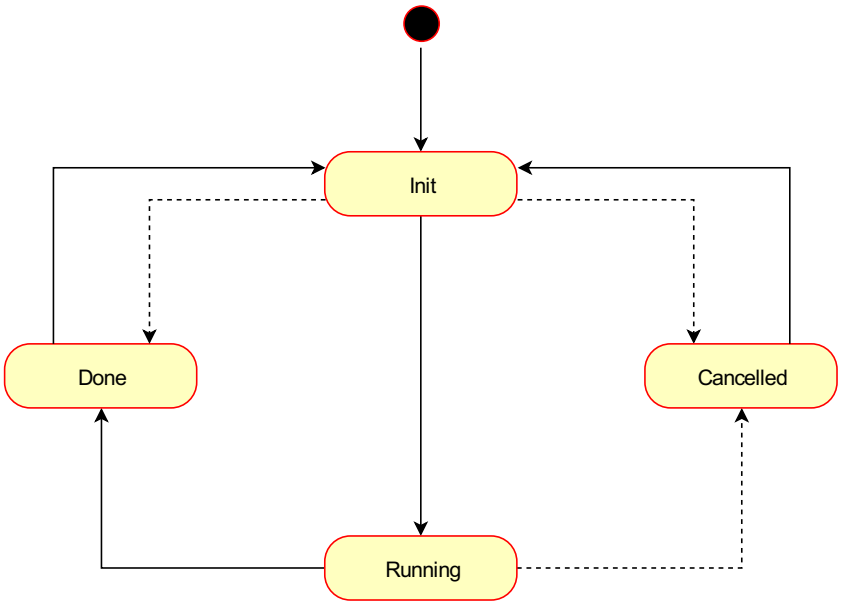
CommandBase

Purpose of this document

Describe the principles of a `CommandBase` Class.

States

The Button has the following default states represented as the `stHandshake.nResultCode` and defined in `E_CommandResultCode`. The following diagram shows in bold lines the regular state flow and in the dashed lines the possible alternative flow.



> It is possible to set `stHandshake.nResultCode` to any `UINT` value for custom purpose. The result code can be set trouth the `Done` Methode.

Methods

See in the TwinCat 3 Library Manager, in the library. Expand the Command folder and then expand also the `CommandBase` Class to show the methods. By selection of each Method it will show also the explanation.

- Call ? The method is called in each cycle to execute the funktional part and the states of the command implementation.
- Init ? Will be executed in the same cycle `stHandshake.bExecute` changes to `true`
- Task ? Execute the funktional part of the command implementation. Will be executed in the same cycle `stHandshake.bExecute` changes to `true` after the `Init` Method
- CalculateProgress ? Can be used to calculate the `stHandshake.nProgress` and `stHandshake.nSubTask`
- Done ? Can be use to finish the `Task` Execution with a `nResultCode`.
- Abort ? Abort the `Task` Execution with the result code `Cancelled`

Properties

See in the TwinCat 3 Library Manager, in the library. Expand the Command folder and then expand also the `CommandBase` Class to show the Properties. By selection of each Property it will show also the explanation.

- Progress ? Set the Progress from everywhere in the application (Regular from CalculateProgress Method)
- SubTask ? Set the SubTask Information from everywhere in the application (Regular from CalculateProgress Method)

Communication structure to PCS

In the structure `ST_CommandBaseHandshake` is used to communicate with the PCS. The flags `bExecute` and `bBusy` has a Impact to command state.

SYMBOL NAME	DATATYPE	PLC	PCS	DESCRIPTION
`bExecute`	BOOL	RW	RW	Will be set to true from the PCS to start the command. When the operation is finished, the PLC will it set to false. On a long running Task, the PCS can reset to false for Abort the operation. The state will change to `Cancelled`.
`bBusy`	BOOL	RW	R	Is True when the command is executing.
`nResultCode`	UINT	RW	R	Shows the operation result code of the commando state. Default Values see `CommandResultCode` Enum Type. It is possible to set other codes!
`nProgress`	BYTE	RW	R	Shows the optional calculated progress. The value is depend on the command implementation. (default can be 0..100%). Only necessary on long running operations.
`nSubTask`	UINT	RW	R	Shows the optional state of long running Command Execution. It is possible to set it on own need.

Values of CommandResultCode:

```
TYPE E_CommandResultCode :
(
    Init            := 0,
    Running         := 1,
    Done            := 2,
    Cancelled       := 3,
    StartCustom     := 100
) UINT;
END_TYPE
```

Quick start with your first Command

Goal: Get the information when the command are executed triggered from the PCS (there is maybe a button). We wanna only a simple impulse when that happen on the output `Q`.

First add a new POU for the Command that extends the Library Class `CommandBase`.

Basely you have a working command that does nothing important. So we add a Output Parameter `Q`. Also a `bInit` for later use.

```
FUNCTION_BLOCK PUBLIC StartCommand EXTENDS CommandBase
VAR_INPUT
END_VAR
VAR_OUTPUT
    Q : BOOL;
END_VAR
VAR
    bInit   : BOOL;
END_VAR
```

Then we ned to now when the Command is executed. For that we add a Method with the Name `Init` to the Class.

```
METHOD PROTECTED Init

// Init Code:
bInit := TRUE;
```

The Next step is to implement to logic Part of the command, to ged the information when the command are executing and to set the the impulse on `Q` when that happens. For this add a Method with the Name `Task` to the Class.

```
METHOD PROTECTED Task : BOOL

// Task Code:
// dedect task completed
IF (NOT bInit AND Q) THEN
    // Task comlete;
    Task := SUPER^.Task();;
END_IF

// set output trigger
Q := bInit;

// Reset after 1. cycle
bInit := FALSE;
```

The next step is to Declare the POU on a Global Variable List, in this example it is on `Commands`:

```
VAR_GLOBAL
    StartCommand1       : StartCommand;
END_VAR
```

In the main Program you should now call the Button in a cycle behavior.

```
Commands.StartCommand1.Call();

fbTofStartCommand1(
    IN := Commands.StartCommand1.Q,
    PT := T#1S);
```

Now there is simple Start Button created.

To Test the correctness of the behavior. Set the following internal `SartCommand` flag `Commands.StartCommand1.stHandshake.bExecute` in the online Scope to True.

Changelog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog and this project adheres to Semantic Versioning.

The following types of changes exist:

- **Added** for new features.
- **Changed** for changes in existing functionality.
- **Deprecated** for soon-to-be removed features.
- **Removed** for now removed features.
- **Fixed** for any bug fixes.
- **Security** in case of vulnerabilities.

[1.1.0] - 25.04.2018

Added

- New Constant in `E_CommandResultCode` which indicates the start of the user result codes
- Properties to set the progress and sub-task frok the Task-Method

[1.0.0] - 18.04.2018

Added

- Created PCS Library
- Added CommandBase class for default PCS communication with .NET over ADS
- Readme and this changelog documentation