BeyondHMI

Tank Supervisor Library

Developer's Guide: using BhiLibTank in an e!COCKPIT project

The following sections provide detailed instructions for creating a simple e!COCKPIT program which uses the tank supervisor library.

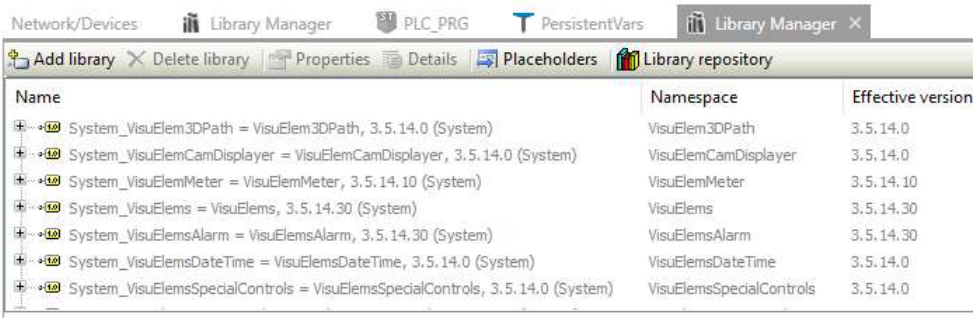
Contents

Obtain the library file

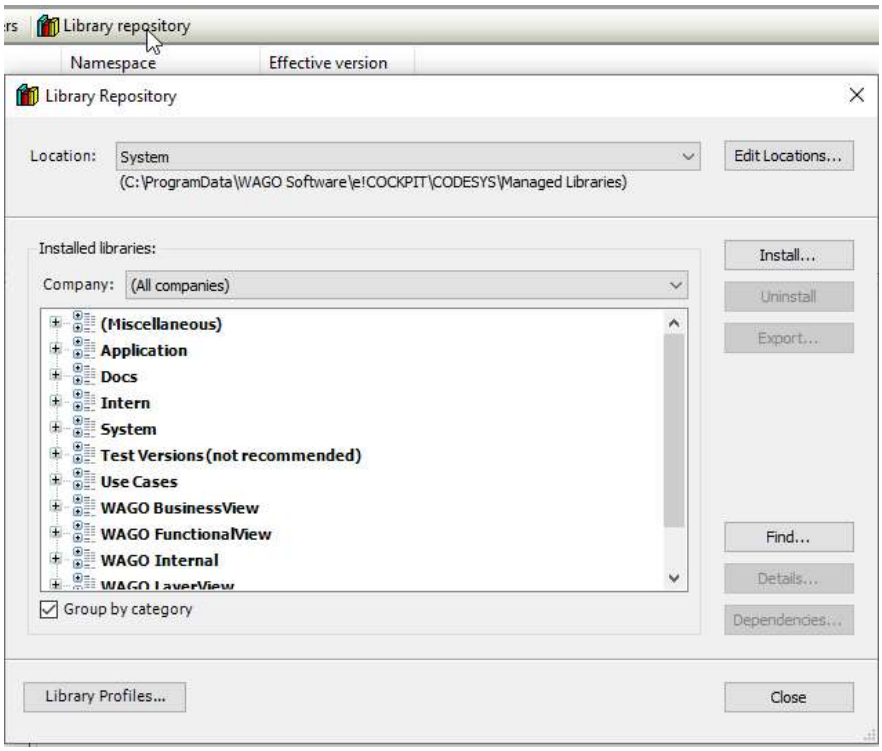
Request the appropriate BhiLibTank.compiled-library from www.Beyond-HMI.com.

Install the library in e!COCKPIT

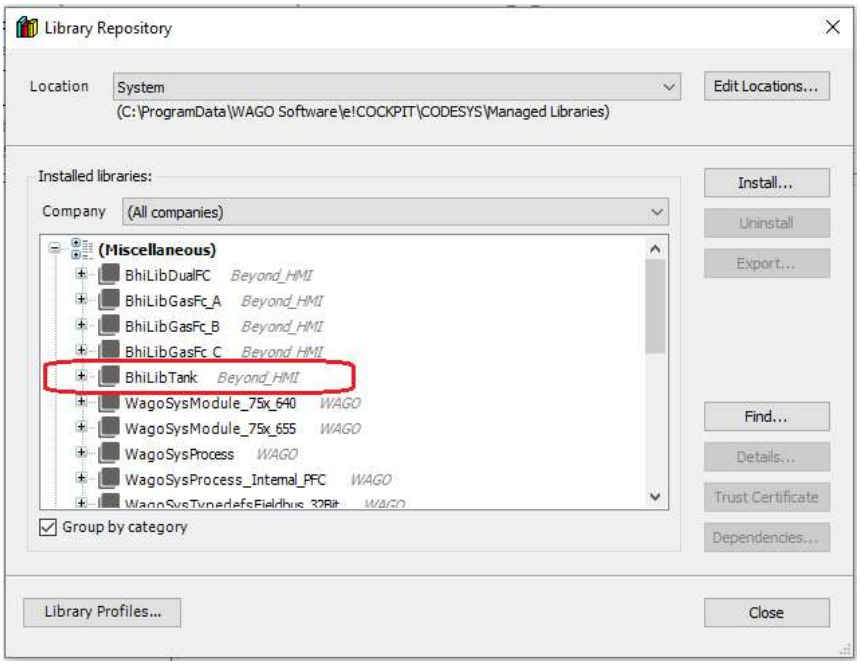
- Open any project in e!COCKPIT
- Navigate to a Library Manager



- Select **Library Repository**

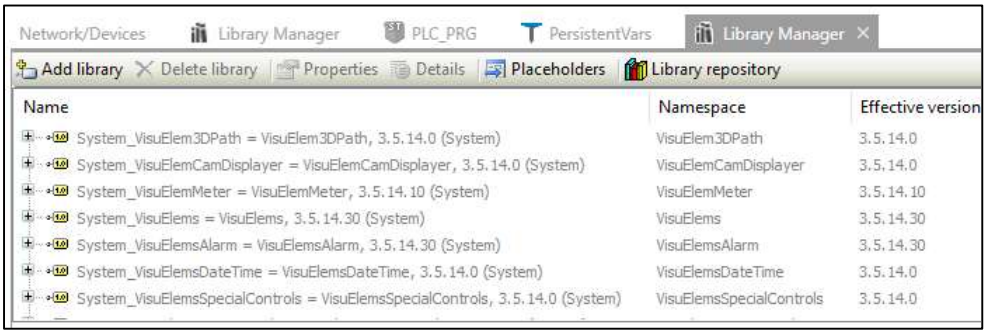


- Select **Install..**
- Navigate to the downloaded library file and click on **Open**.
- Verify that the library was installed in the **Miscellaneous** section

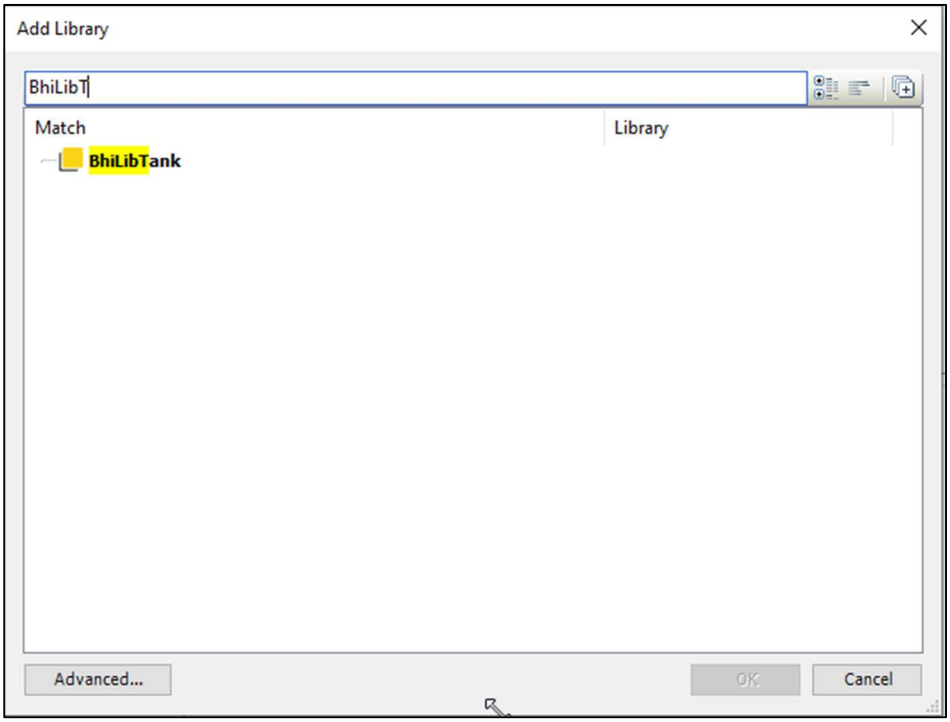


Add the Library to Your Project

- Create a project and designate the Device(s) in the project.
- Navigate to a Library Manager



Select **Add Library**



Start typing the library name until the library appears in bold text

Select the Library and select **OK**.

Specify how much memory will be needed for your implementation

The BhiLibTank library can support from 1 to 254 tanks and from 1 to 254 pump controllers. Each one of these entities occupies a certain amount of persistent memory and a certain amount of volatile memory. For efficient memory utilization, the host PLC program (your program) must specify the number of instances of each entity type. Select these quantities carefully – so PLC memory is not wasted.

Also, depending upon how you will use strapping tables and how many strapping table you will use, you will need to specify some quantities to indicate ho much memory the library will allocate for tank strapping table data.

Good coding practices suggest that these quantities should be specified as global constants. However, this is not strictly required. The code sample below shows how these global constants might be declared.

```
/{attribute 'qualified_only'}
VAR_GLOBAL CONSTANT

      NUM_TANKS : BYTE := 3;
      NUM_PUMP_CONTROLLERS : BYTE := 2;
      NUM_STRAPPING_TABLES : BYTE := 2;
      NUM_STRAPPING_DETAILS : UINT := 1024;
END_VAR
```

The table below provides some description of the constants declared in the sample above and their usage:

Variable	Description	Valid Range
NUM_TANKS	Maximum number of tanks to monitor	0 thru 254 (do not use 255)
NUM_PUMP_CONTROLLERS	Maximum number of pump controllers to process	0 thru 254 (do not use 255)
NUM_STRAPPING_TABLES	Maximum number of strapping tables which will be used in the installation	0 thru 10. A limit of 10 strapping tables is hard-coded in the library.
NUM_STRAPPING_DETAILS	Total number of strapping table rows (combined total of all strapping tables)*	0 through 65534

A strapping table "row" is a pair of (tank height, tank volume). Regardless of number of strapping tables used, all "rows" are stored in a common list. For example, if you have one table with only 10 rows and second table with 300 rows and a third table with 400 rows, you only need to allocate a total of 710 rows. It is good practice to round up, so the recommended setting for this case would be "`NUM_STRAPPING_DETAILS : UINT := 1024;`".

Create persistent memory structures which will be used by the library

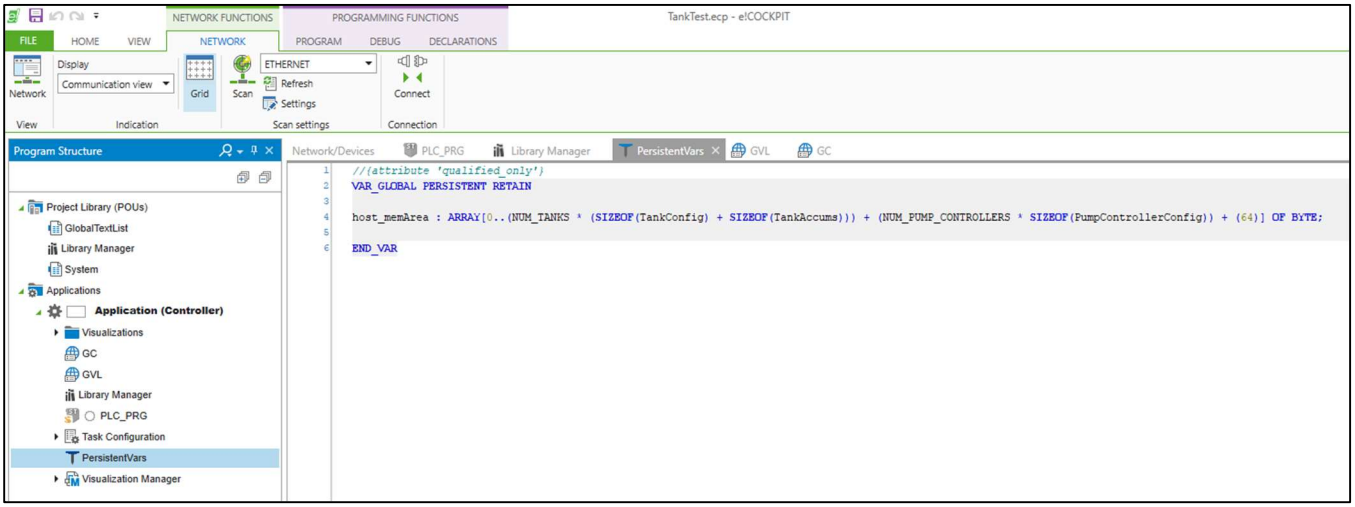
The library needs some of its data structures to persist – even when the PLC program is loaded or the power to the PLC is cycled. Your program needs to allocated memory which will hold these structures and will also be maintained by the PLC in a persistent state.

If one has not already been created, add a **Persistent Variables** Object to the Project

Add the following declarations to the persistent memory area (Copy these lines into the e!COCKPIT window):

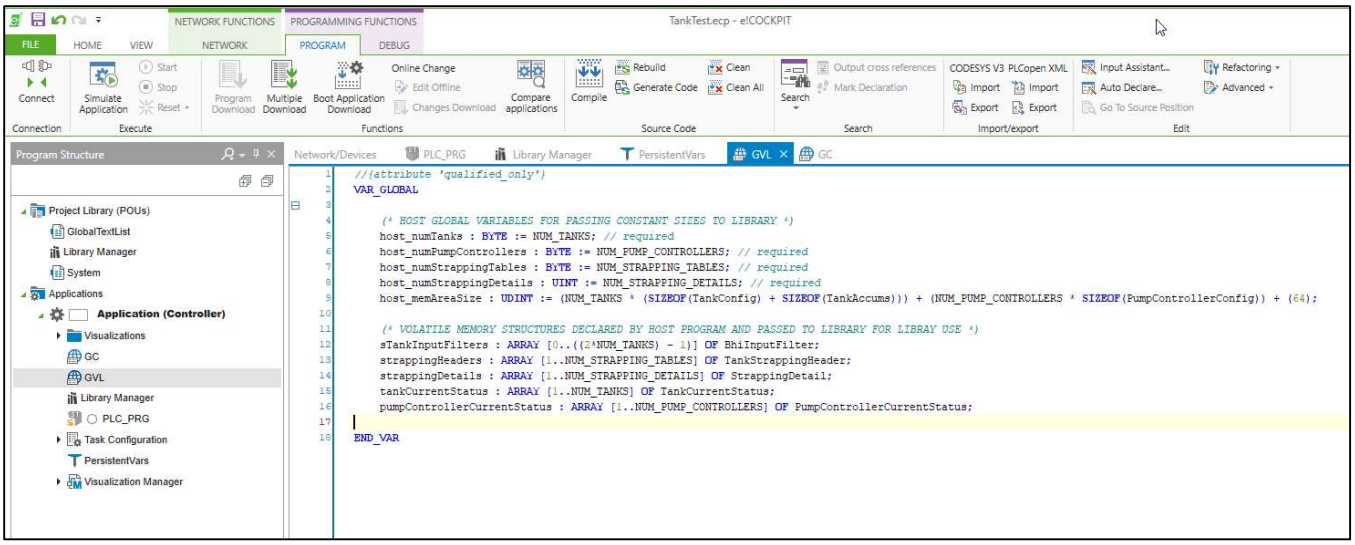
```
VAR_GLOBAL PERSISTENT RETAIN

host_memArea : ARRAY[0..(NUM_TANKS * (SIZEOF(TankConfig) + SIZEOF(TankAccums))) +
(NUM_PUMP_CONTROLLERS * SIZEOF(PumpControllerConfig)) + (64)] OF BYTE;
END_VAR
```

Note that in the example above, global constants are used for some of the quantities. See the previous section for details about the meaning of these constants. Literal number could be used – if desired.



Add necessary variables to the Program which calls the library function block

The following global variables must be declared:




```
//{attribute 'qualified_only'}
VAR_GLOBAL

      (* HOST GLOBAL VARIABLES FOR PASSING CONSTANT SIZES TO LIBRARY *)
      host_numTanks : BYTE := NUM_TANKS; // required
      host_numPumpControllers : BYTE := NUM_PUMP_CONTROLLERS; // required
      host_numStrappingTables : BYTE := NUM_STRAPPING_TABLES; // required
      host_numStrappingDetails : UINT := NUM_STRAPPING_DETAILS; // required
      host_memAreaSize : UDINT := (NUM_TANKS * (SIZEOF(TankConfig) +
SIZEOF(TankAccums))) + (NUM_PUMP_CONTROLLERS * SIZEOF(PumpControllerConfig)) +
(64);

      (* VOLATILE MEMORY STRUCTURES DECLARED BY HOST PROGRAM AND PASSED TO LIBRARY
FOR LIBRAY USE *)
      sTankInputFilters : ARRAY [0..((2*NUM_TANKS) - 1)] OF BhiInputFilter;
```

```
      strappingHeaders : ARRAY [1..NUM_STRAPPING_TABLES] OF TankStrappingHeader;
      strappingDetails : ARRAY [1..NUM_STRAPPING_DETAILS] OF StrappingDetail;
      tankCurrentStatus : ARRAY [1..NUM_TANKS] OF TankCurrentStatus;
      pumpControllerCurrentStatus : ARRAY [1..NUM_PUMP_CONTROLLERS] OF
PumpControllerCurrentStatus;

END_VAR
```

Add supporting Code to the Program

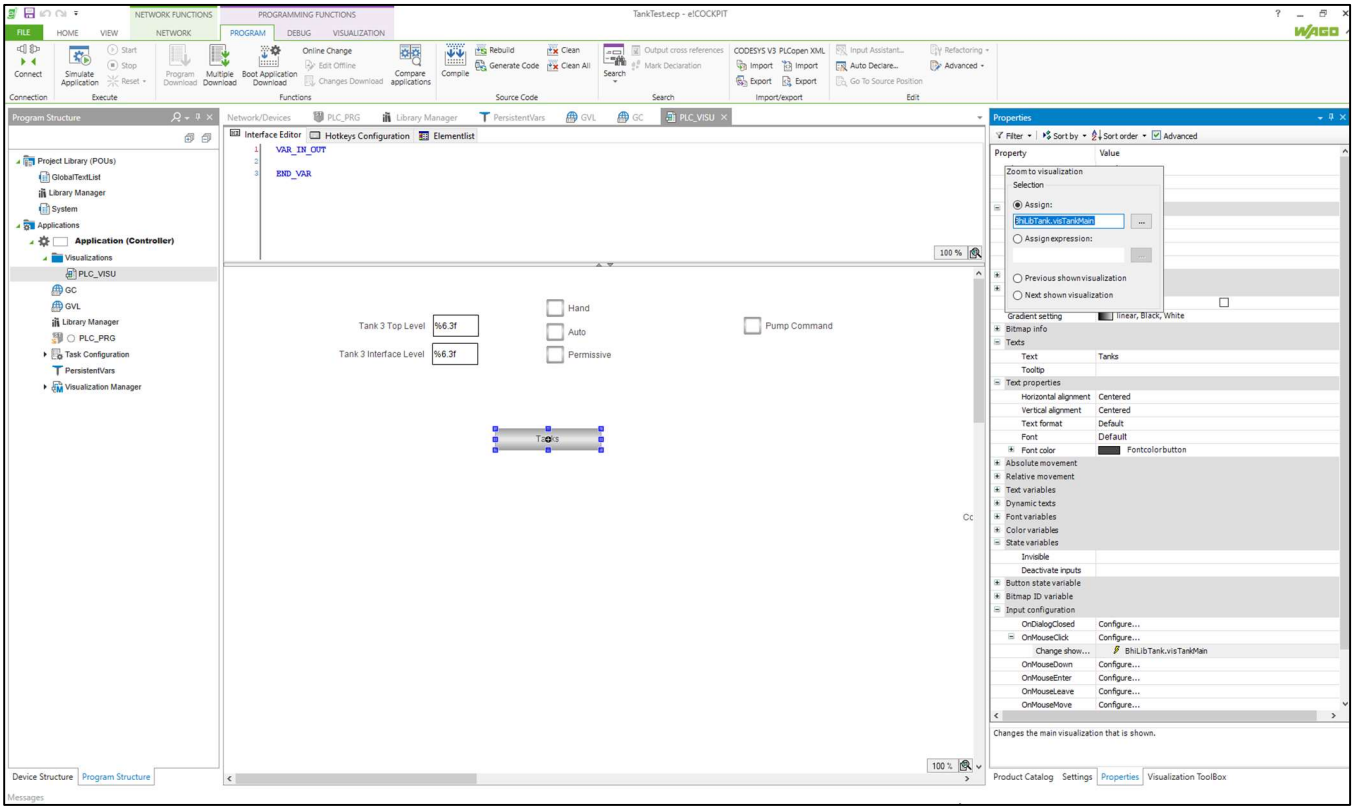
The following illustrates how the library can be used with a Structured Text Program.

```
PROGRAM PLC_PRG
VAR
      ////////////////////////////////////////////////////////////////////
      // THIS VARIABLE IS USED TO CONTROL THE FLOW OF THE MAIN PROGRAM
      // ON FIRST TASK CYCLE, CERTAIN INITIALIZATION/HOUSEKEEPING TASKS MUST BE
DONE - JUST ONCE
      ////////////////////////////////////////////////////////////////////
      xIsInitialized : BOOL := FALSE;

      ////////////////////////////////////////////////////////////////////
      // YOU MUST DECLARE AN INSTANCE OF THE TankTwin FUNCTION BLOCK
      ////////////////////////////////////////////////////////////////////

      theTankManager : TankTwin;


END_VAR

IF (xIsInitialized) THEN

      ////////////////////////////////////////////////////////////////////////////
//
      // AFTER INITIALIZATION (one time), PERFORM THIS CYCLE OF CALLS TO THE
LIBRARY
      ////////////////////////////////////////////////////////////////////////////
/

      // for each tank, your code needs to interact with the I/O or network and get
the latest values from tank gaugers
      // the example below shows how to update the library for tank 3 with the
latest values (tanks are "one indexed")
      // this code assumes that your program (elsewhere) is updating the variables
rTank3Top and rTank3Interface
      // NOTE THAT THE UPDATE CALL ONLY NEEDS TO BE EXECUTED WHEN THE "TOP" AND/OR
"INTERFACE" LEVELS HAVE BEEN UPDATED BY THE GAUGER (EG: WIRELESS)
      // HOWEVER, IT IS OK TO MAKE THIS CALL ON EVERY SCAN - USING THE SAME VALUE
MULTIPLE TIMES - UNTIL THE GAUGER VALUE IS UPDATED
      theTankManager.UpdateTank(3,rTank3Top,rTank3Interface);

      // call the tank manager block on each scan
      theTankmanager();
```

```
      // if you are using pump controllers, call the "UpdatePumpController" method
on each scan for each pump controller
      // the example below shows how to update pump controller number 1
(pumpcontrollers are "one indexed")
      // this code assumes that your program (elsewhere) is interacting with the
I/O or network to obtain the most recent status of the HOA switch and permissive
      // NOTE: the return value from the "UpdatePumpController" method is a BOOLEAN
- indicating whether the pump should be running or not
      xPumpCommand :=
theTankManager.UpdatePumpController(1,xHandState,xAutoState,xPermissiveState);


ELSE
      // the first call made to the library should be to the "FuInitialize" method
      // in this call, you pass size parameters and pointers to the memory that
your program allocated for library use
      theTankmanager.FuInitialize(ADR(host_memArea),
                                                host_memAreaSize,
                                                host_numTanks,
                                                host_numPumpControllers,
                                                ADR(sTankInputFilters),
                                                TRUE, // boolean indicating whether
to use tank gauger filtering (applies to all tanks)
                                                host_numStrappingTables,
                                                ADR(strappingHeaders),
                                                host_numStrappingDetails,
                                                ADR(strappingDetails),
                                                ADR(tankCurrentStatus),
                                                ADR(pumpControllerCurrentStatus));

      xIsInitialized := TRUE;
END_IF
```

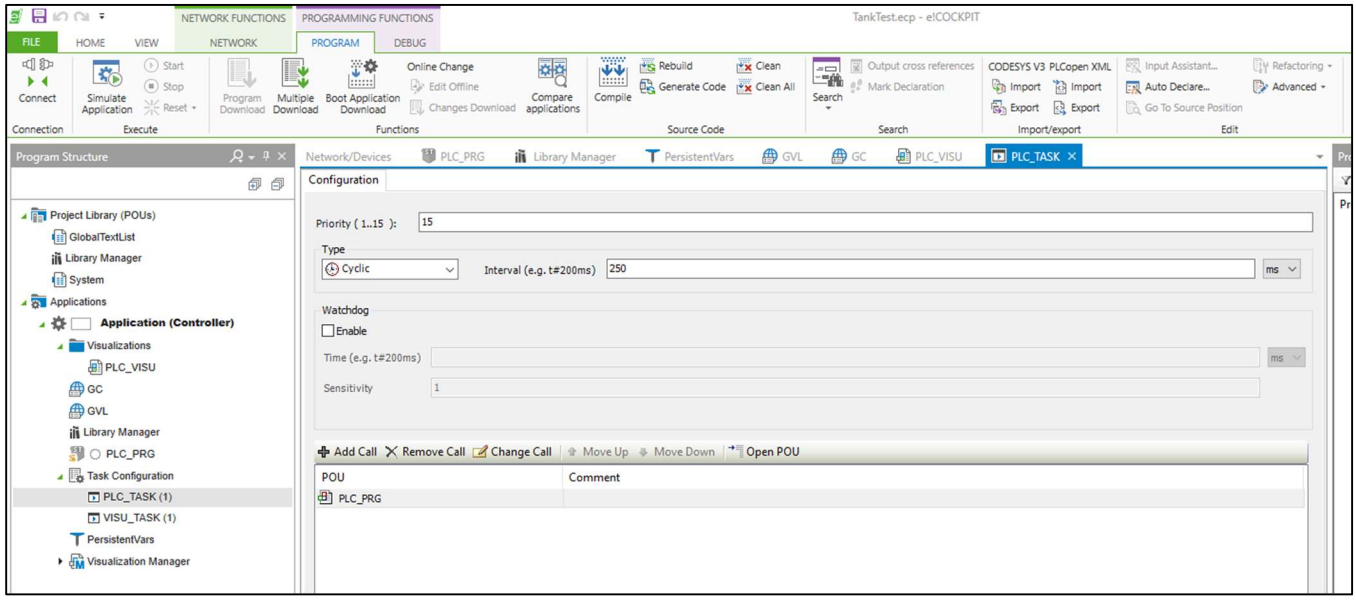
Link Program Visualizations to Library Visualizations

The library includes a number of visualizations for interacting with the Tank and Pump Controller functions. The library visualizations can all be accessed from BhiLibTank.visTankMain. The screen capture below shows an example of a simple visualization containing a single button linking to the flow computer menu Visu.



Adjust e!COCKPIT project Task Interval

The library needs to be called should be called at least every 500 milliseconds. If less than 10 tanks are being monitored or if the tank gauger values are updating infrequently (e.g. wirelessly), then 500 msec scan rate is adequate. For installations with significantly higher tank counts, a scan frequency of 250 msec or even 100 msec might make the library more responsive. There is not really any good reason to scan faster than 100 msec. You can adjust the scan frequency of this task to manage PLC CPU loading.


NETWORK FUNCTIONS
PROGRAMMING FUNCTIONS
TankTest.ecp - e!COCKPIT
FILE
HOME
VIEW
NETWORK
PROGRAM
DEBUG
Connect
Simulate Application
Start
Stop
Reset
Program Download
Multiple Download
Boot Application Download
Online Change
Edit Offline
Changes Download
Compare applications
Compile
Rebuild
Generate Code
Clean
Clean All
Search
Output cross references
Mark Declaration
CODESYS V3 PLCopen XML
Import
Export
Import
Export
Input Assistant...
Auto Declare...
Go To Source Position
Refactoring
Advanced
Connection
Execute
Functions
Source Code
Search
Import/export
Edit
Program Structure
Project Library (POUs)
GlobalTextList
Library Manager
System
Applications
Application (Controller)
Visualizations
PLC_VISU
GC
GVL
Library Manager
PLC_PRG
Task Configuration
PLC_TASK (1)
VISU_TASK (1)
PersistentVars
Visualization Manager
Network/Devices
PLC_PRG
Library Manager
PersistentVars
GVL
GC
PLC_VISU
PLC_TASK
Configuration
Priority (1..15):
15
Type
Cyclic
Interval (e.g. t#200ms)
250
ms
Watchdog
Enable
Time (e.g. t#200ms)
Sensitivity
1
Add Call
Remove Call
Change Call
Move Up
Move Down
Open POU
POU
Comment
PLC_PRG

Licensing

The BhiLibTank library utilizes runtime licensing. Each PLC upon which it runs must have a license. Licenses ae obtained from Beyond HMI, Inc.

Trial Mode

Upon startup, the library will run in trial mode for approximately 4 days. While in trial mode, the library is fully functional. After the 4 day period passes – and if no license is installed - the library will stop calculating.

If the PLC program is stopped and restarted, the 4 day trail period begins again. Therefore, PLC program developers should be able to develop and test programs without needing a license for their development PLCs.

Steps to Obtain a Runtime License

To fully license the BhiLibTank library on a PLC, the following steps must be executed:

- Include library features in a PLC program (*reference other instructions for PLC program developers within in this document*)
- Install the PLC program on the target PLC specimen
- Open the library's Admin screen and capture the Site Code
- Transmit the site code to Beyond HMI, Inc. and provide payment information
    - Please use info@beyond-hmi.com to initiate contact with us.
- Wait for Beyond HMI, Inc. to return a license file
- Install the license file (using Beyond HMI's free software "BLT")
- Open the library's Admin screen and confirm that the license check result is green

Licenses are perpetual. No maintenance fee is required. Licenses are keyed to a site code and are not portable between PLCs. Please contact Beyond HMI if you need to move a license to another PLC.

Modifying your PLC program without corrupting library data

Your PLC program will inevitably need to be modified – possibly after library configuration has been done and loadouts have been accumulated by the library. Certain changes to your PLC program (adding persistent variables, for example) can cause data in the library's memory space to be cleared or corrupted. Beyond HMI has developed tools to support changing your PLC program without losing persistent library data. The following section describes the procedure you should follow to maintain the integrity of your library data while making PLC programming changes:

Note: The following steps must be executed in order. Please read and study the entire procedure list before beginning PLC program maintenance.

Save a maintenance file

Check the **Save Maintenance File** checkbox on the Advanced Admin screen. Wait for the checkbox to be unchecked. This indicates that a maintenance file has been saved to the PLC file system.

Perform PLC program maintenance

At this point, you are free to make changes to the PLC program and load those changes onto the PLC.

Force Maintenance Recovery

Check the **Force Maintenance Recovery** checkbox on the Advanced Admin screen. Wait for the checkbox to be unchecked. This indicates that meter accumulators have been recovered from the maintenance file.

How your program can interact with the Library

In addition to the requirements of initializing the library and passing live meter readings to the library, your program code can interact with the BhiLibTank library to:

- Read current accumulation of tank loadout quantities (daily, weekly)
- Read current tank height and inventory information
- Read current pump controller status information
- Read and Write tank configuration parameters
- Read and Write pump controller configuration parameters

The following sections provide further detail about how to execute these interactions from your program code.

Reading Current Accumulated Loadout Quantities from your program

Use the *TankTwin.GetTankLoadoutStats* method to accumulated tank loadout volumes.

This method takes a single input parameter:

- tank number (1 for the tank, etc.)

And four input_output (by reference) parameters:

- Barrels today [UDINT]
- Barrels yesterday [UDINT]
- Barrels this Month [UDINT]
- Barrels Previous Month [UDINT]

Reading Current Tank height and inventory information

To obtain tank status information, simply read from the tankCurrentStatus array in your program. This array is a “one-indexed” array of TankCurrentStatus structures.

For instance, to read the current values for filtered tank top in feet, inches, and 18ths of an inch, for the second tank, read:

- tankCurrentStatus[2].sTopFeetAndInches.iAs8ths_Feet
- tankCurrentStatus[2].sTopFeetAndInches.iAs8ths_Inch
- tankCurrentStatus[2].sTopFeetAndInches.iAs8ths_8ths

Reading Current Pump Controller status information

To obtain pump controller status information, simply read from the pumpControllerCurrentStatus array in your program. This array is a “one-indexed” array of PumpControllerCurrentStatus structures.

For instance, to read the current state of the commanded output from the first Pump Controller, read:

- pumpControllerCurrentStatus[2].booleanAttributes[OUTPUT_VALUE]

Reading and Writing Tank configuration parameters

To read tank configuration parameters from the library, use the *TankTwin.GetTankConfig* method:

- Declare a variable of type TankConfig
- Call the GetTankConfig method on the TankTwin Function Block
- If the return value from the value from this call is TRUE, then read the parameter(s) of interest

For instance, to read the height of the first tank instance:

```
myTankStruct : TankConfig;

IF (theTankTwin.GetTankConfig(1, myTankStruct)) THEN

      myVariable := myTankStruct. rTankHeight;

END_IF;
```

To modify configuration parameters in the library, use the *TankTwin.GetTankConfig* method to get the current configuration, then modify the desired parameters, then use *TankTwin.SetTankConfig* to set the library values.

- Declare a variable of type TankConfig
- Call the GetTankConfig method on the TankTwin Function Block
- If the return value from the value from this call is FALES, then there was an error
- Otherwise,
    - modify the parameters in the TankConfig struct
    - Call the SetTankConfig method

For instance, to modify the height of the first tank instance:

```
myTankStruct : TankConfig;

IF (theTankTwin.GetTankConfig(1, myTankStruct)) THEN

      myTankStruct. rTankHeight := <new value>;

      theTankTwin.SetTankConfig(1, myTankStruct);

END_IF;
```

Reading and Writing Pump Controller configuration parameters

To read pump controller configuration parameters from the library, use the *TankTwin.GetPumpControllerConfig* method:

- Declare a variable of type PumpControllerConfig
- Call the GetPumpControllerConfig method on the TankTwin Function Block

- If the return value from the value from this call is TRUE, then read the parameter(s) of interest

For instance, to read the "pump on" high level for the first pump controller instance:

```
myPumpControllerStruct : PumpControllerConfig;
IF (theTankTwin.GetPumpControllerConfig(1, myPumpControllerStruct)) THEN
      myVariable := myPumpControllerStruct. rTankOnLevelSetpoint;
END_IF;
```

To modify configuration parameters in the library, use the *TankTwin. GetPumpControllerConfig* method to get the current configuration, then modify the desired parameters, then use *TankTwin. SetPumpControllerConfig* to set the library values.

- Declare a variable of type PumpControllerConfig
- Call the GetPumpControllerConfig method on the TankTwin Function Block
- Otherwise,
    - modify the parameters in the PumpControllerConfig struct
    - Call the SetPumpControllerConfig method

For instance, to modify the "pump on" high level for the first pump controller instance:

```
myPumpControllerStruct : PumpControllerConfig;
IF (theTankTwin.GetPumpControllerConfig(1, myPumpControllerStruct)) THEN
      myPumpControllerStruct. rTankOnLevelSetpoint := <new value>;
      theTankTwin.SetPumpControllerConfig(1, myPumpControllerStruct);
END_IF;
```