# LLSYMBOLS MANUAL

**Technical specification n° SP18001**

| Version | Author | Date | Note |
|---|---|---|---|
| | L. Bollini | 25/07/2018 | First release |
| 3.0.0.0 | M. Fomasi | 11/04/2019 | Groups, versioning and validation |

# Table of contents

# 1 INTRODUCTION

LLSymbols is a software library, available both on Windows and Linux environments, that exports some useful functions which allow any external 3rd party software to manipulate symbols (which are PLC variables declared inside your own PLC program) with various queries and read/write operations.

# 2 REQUIREMENTS

LLSymbols is made up of a client side (LLSymbols.dll / LLSymbols.so or others depending on the specific operating system) and a server that must always exist along with the PLC runtime on the target. LLExec always comes with it on the target.

Please be sure that the LogicLab version you got downloads the Symbol Table along with the PLC.

This library requires a reference to a given symbol that belongs to the symbol table, that is an XML formatted file containing all the symbols declared into your own specific PLC program.

# 3 ASSETS

The library uses specific assets that are created on the target side and accessed by the client through a handle object.

It is very important that you remember to close these handles when you don't need them anymore, since they consume resources on the target.

When you download a new PLC program, any handle previously used gets invalidated and, for this reason, the client must close and reopen them all.

In practical terms, LLSymbols formalizes this into a pair of concepts that are a workspace and a global tag.

## 3.1 WORKSPACE

A workspace is an application domain for a group of symbols, since it allows you to create a connection to a PLC and to access or even modify the values related to its tags.

## 3.2 TAG

A tag is an object that represents a given symbol within that workspace. Please be aware that a tag cannot exist if not inside a given workspace.

## 3.3 TYPE

It is also possible to get a type handle object that can be either a base or a complex type, and in this last case you can define arrays, matrices (up to 3 dimensions), structs or function blocks.

### 3.4 GROUP

A group is a collection of tags. It is possible to create more groups with no restrictions about the types of the tags of the group.
Using groups it is possible to read all the values of the tags of a group from the PLC server or write all the values of the tags of a group into the PLC by calling using only one call.
Using groups optimize read/write operations.

## 4 GENERAL USAGE NOTES

All the functions return a code (LLSymResult) that represents any kind of error that can take place during the execution. In case a function correctly exits with code 0, that means no errors did occur.

### 4.1 SUPPORTED COMMUNICATION PROTOCOLS

There are two kinds of communication protocols supported by this library:

- GDB on a TCP connection;
- Commlibs (on Windows).

### 4.2 SUPPORTED LANGUAGES

Supported languages are:

- C (take a look at LLSymbols.h);
- All .NET languages are supported through the importation of native functions (please take a look at LLSymbolsVB.NET\LLSymbols.vb);
- LLSymbols is distributed as both a static and dynamic library, either on Windows and Linux environments.

### 4.3 LLSYMBOLS CLIENT / SERVER COMPATIBILITY

LLSymbols API version makes requests to a server. Depending on the version of the symbols server running on the target, connection and communication can be fully supported, partially supported or avoided.
LLSymbols is versioned in this way: Major.Minor.Revision.Build (ie: 3.0.0.0)
LLSymbols version is checked during LLSym_OpenWorkspace. This call returns one of the following results:

- symResOk
  LLSymbols API client has access to all server features.
  Same major version, server minor version or revision are greater than or equal to client version.

- symResOkNotFullySupported:
  LLSymbols API client can connect to symbols server but not all requests are supported by server. LLSymbols client version is newer than server
  Same major version, server minor version or revision are lower than client version.

- symResServerNotCompatible:
  Client is not compatible with server.
  Major version mismatch.

- Other error codes indicate connection error or incompatible versions

### 4.4 WORKSPACE VALIDATION

Workspace is strictly related to a symbol table that describes all the PLC symbols.

This relation set when a new workspace is created between workspace, symbol table and PLC must be checked to grant data integrity.

At runtime it is possible that PLC and symbol table are changed by PLC developer. In this case existing workspaces are considered no more valid.

Client requests are always validated by server to avoid inconsistent operations.

If the handle (workspace, tag, type, group) specified by client in a request is no more valid because PLC changed symResInvalidHandle is returned.

Explicit check to be sure worspace is ok can be done calling LLSym_IsValidWorkspace.

If workspace is no more valid, a new workspace must be opened by client.

### 4.5 FUNCTION TYPES

#### 4.5.1 READ / WRITE FUNCTIONS

There are several ways these functions can be performed:

- using a buffer;
- using VARIANT (on Windows only);
- on a single element or with a full read/write of arrays and structures elements as blob binaries.

#### 4.5.2 QUERY-BASED FUNCTIONS

These functions return information either on the PLC variables or on enumerator types, elements and structures.

## 5 USAGE EXAMPLE

The first thing you should do consists in work by invoking the initialization function. Then generate a new workspace handle and finally get a global tag handle for a symbol you want to manipulate. Once you got this tag, you can use it as an argument of the functions that perform the read/write operations for its related value.

If you decide to close an instance of workspace, first of all you should close all its related global tags that you had previously opened, since they should not exist anymore. After that, you must close the workspace and shutdown the system.

A typical procedure consists of the following steps:

- Call the initialization function (LLSym_Init);
- Open a workspace;
- Open a tag;
- Read or write a value for that tag;
- Close that tag;
- Close the workspace;
- Shutdown the system.

The following lines give you a use case example on how you should use this library.

```
LLSymWorkspaceHandle wks = 0;
LLSymTagHandle tag = 0;
LLSymResult result;
const LLSymServerAddress GDBConn =
               { symProtGDB_TCP, 0, { _T("192.168.100.50"), 5000 } };

result = LLSym_Init();
result = LLSym_OpenWorkspace(0, &GDBConn, &wks);
result = LLSym_GetGlobalTag(wks, _T("COUNTER"), &tag);

value = 11;
result = LLSym_SetTagValue_Ptr(tag, &value, sizeof(value));
result = LLSym_GetTagValue_Ptr(tag, &value, sizeof(value));


result = LLSym_CloseTag(tag);

result = LLSym_CloseWorkspace(wks);
result = LLSym_Shutdown();
```