

LabVIEW

NI LabVIEW is a software for systems design which uses a graphical language, named "G" (not to be confused with the more pleasant G-point), to build complex laboratory and automation applications.

In a G program, the execution is determined by the structure of a graphical block diagram (the LV-source code) on which the programmer connects different function-nodes by drawing wires. These wires propagate variables and any node can execute as soon as all its input data become available.

LabVIEW offers the same data types/structures as other programming languages but they are not "exposed". You know that a cluster (a struct) contains some elements, but you don't know where in the memory they are, i.e. you don't know their physical address.

From this point of view we can consider G as a **managed language**.

Let's see how can we interface LabVIEW with Snap7, keeping in mind these two major differences (execution and data storage) against the traditional programming languages.

The wrapper provided consists of:

1. A LabVIEW library (Snap7.lvlib) that contains a set of Vis. Each vi "wraps" a Snap7 function via the **Call Library function node**.
2. A "glue" DLL (lv_snap7.dll) that interfaces the Vis with Snap7.dll. It re-exports the typed-data functions and supplies new data adapter procedures for the untyped-data functions.

Since many of Snap7 functions only make sense only in a procedural context :

Ø **Asynchronous functions**

All asynchronous functions are not exported, because are completely useless, indeed, in some cases, they can be harmful. LabVIEW is an inherently concurrent, adding a synchronization layer will complicate uselessly the execution flow.

Ø **Callbacks**

LabVIEW cannot natively pass a pointer to a VI for use as a callback function in a DLL, a C wrapper must be used as workaround to provide an interface between the DLL and an user event. This is not a trivial task due to the data-driven nature of the language.

At the end, the Event Structure must be used in a While Loop because when the Event Structure executes, it will only wait for and handle exactly one event.

The Snap7 polling functions must be used instead. This is, imho, the better solution, because they are simple to use and, above all, because LabVIEW has very efficient mechanisms to optimize the parallel executions.

Moreover the polling functions simply check a memory flag, their execution is very fast.

DLL Calling

To understand LabVIEW Snap7 interface, it's important to know how the Call Library function node works.

This is not a commercial book, rewriting base concepts that are already well explained has not much sense. So, to explain this argument, I selected these two pages :

<http://www.ni.com/white-paper/4877/en/>

<https://decibel.ni.com/content/docs/DOC-9080>

As you can see in them, LabVIEW provides two ways to pass complex data to a DLL:

1. Adapt To Type
2. String (as LabVIEW string handle).

The first method is used when the data structure is well known in advance, i.e. when we wire it to the call library node.

All Snap7 functions which declare a struct (in snap7.h) as input use this method.

The second method allows to write VIs that accept, as input, generic buffers encapsulated in a string.

All Snap7 VIs that read/write an untyped buffer use the second method and the data adaption is made in lv_snap7.dll. The string type, in spite of its name, can contain anything since it has in head its length.

Generic buffers

Let see how Snap7 vi manage untyped buffers examining as example the **Cli_DBGet()** function of the client.

This is the C prototype of the function as exported by snap7.dll.

```
int S7API Cli_DBGet(S7Object Client, int DBNumber, void *pUsrData,
    int *Size);
```

This function reads an entire DB of given Number into the buffer pointed by pUsrData.

The first two parameters are simple to manage since they are simple typed vars.

pUsrData is the pointer to a generic buffer.

Size, in input must contain the size of the buffer supplied, in output contains the DB size, i.e how many bytes were read. If the buffer size is less than the DB size an error is returned (but however the buffer contains the partial data read).

The adapter function exported by lv_snap7.dll has this prototype:

```
int S7API lv_Cli_DBGet(S7Object Client, int DBNumber,
    PLVString *pStringData, int *SizeGet);
```

The first two parameters are the same of Cli_DBGet().

PLVString is defined as follow:

```
typedef struct {
    int32_t size;    // Block Size
    byte Data[1];   // Data
}
```

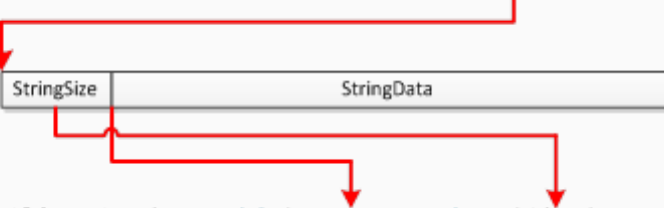
(byte is a portable 32/64 bit "byte" defined in snap7.h).

This is the body of the adapter.

```
int S7API lv_Cli_DBGet(S7Object Client, int DBNumber,
    PLVString *pStringData, int &SizeGet)
{
    int32_t Size = *pint32_t(*pStringData); // String size
    pbyte pUsrData=pbyte(*pStringData) + sizeof(int32_t);
    int Result=Cli_DBGet(Client, DBNumber, pUsrData, &Size);
    SizeGet=Size;
    return Result;
// Note : the buffer size check is performed into Cli_DBGet
}
```

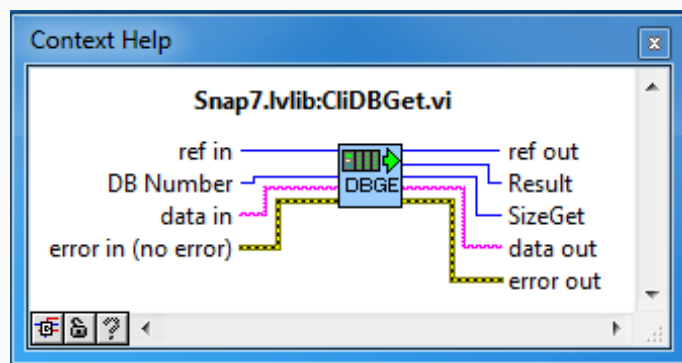
That implements this concept:

```
lv_Cli_DBGet(...PLVString *pStringData,...);
```

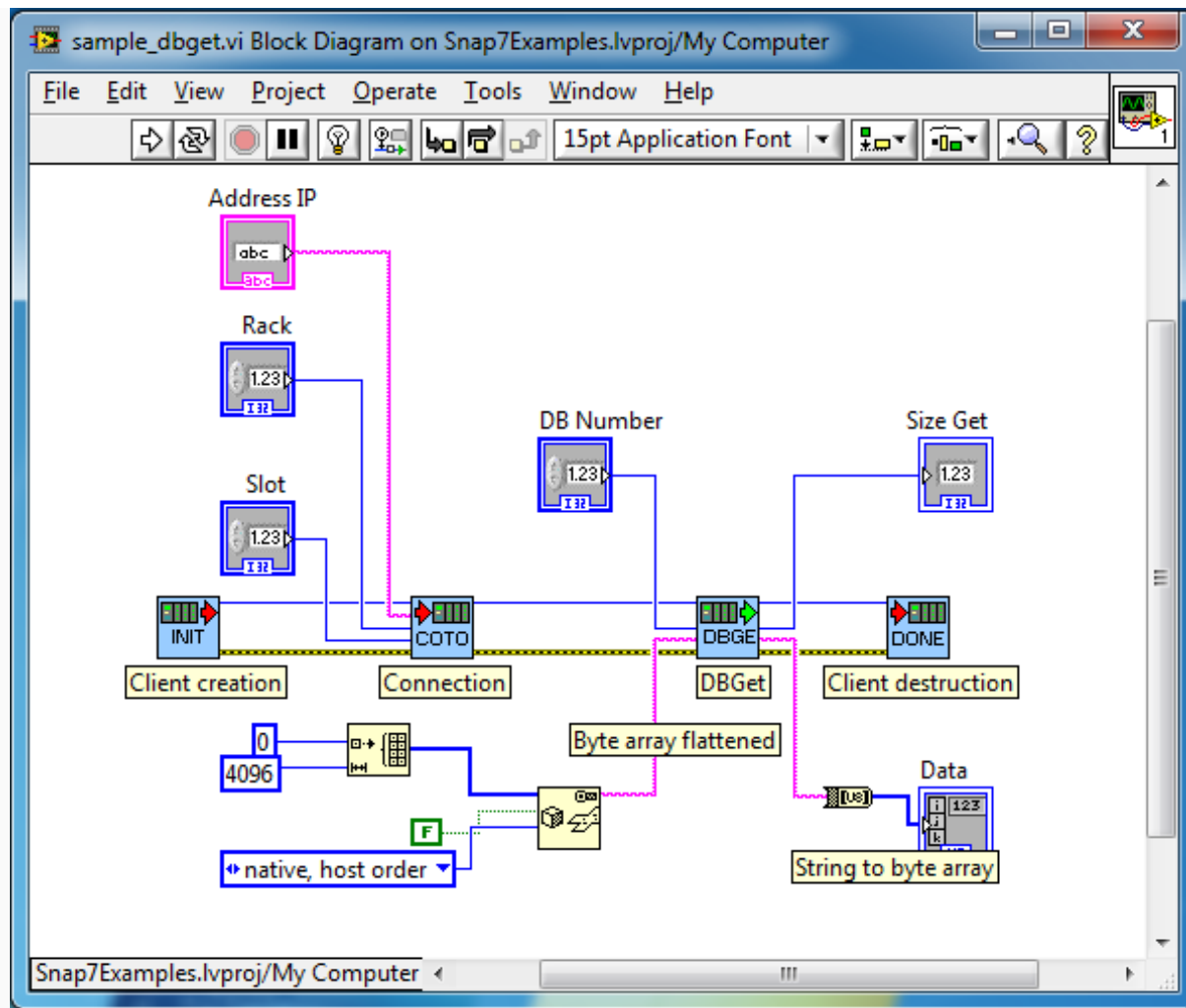


```
Cli_DBGet(...void *pUsrData, int *Size);
```

On the LabVIEW side the vi **CliDBGet.vi** is defined as follow:



Finally, a very minimalist (but working) program to read a DB in a 4K buffer :

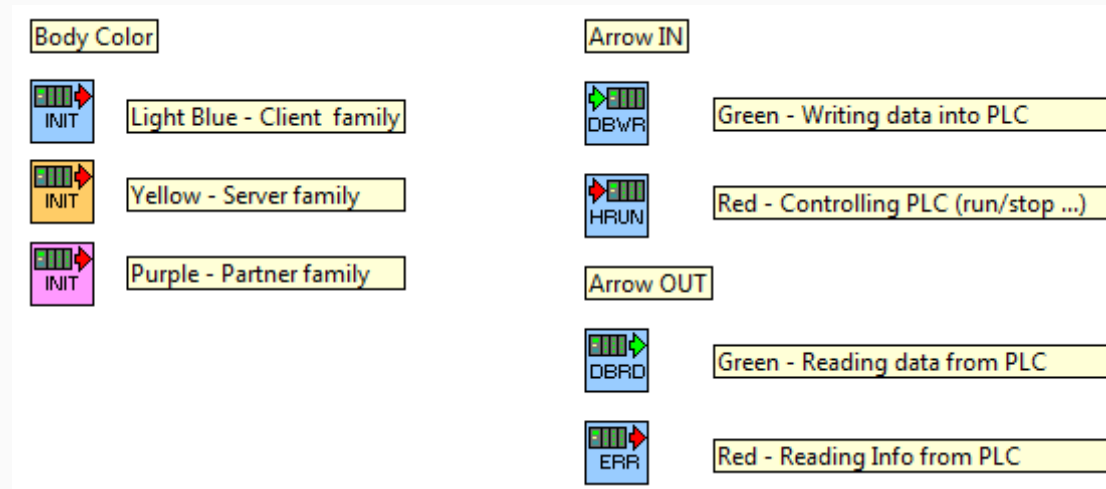


The upper wire across all blocks is the Client reference generated by CliCreate, internally it's a intptr_t, externally is stored into a 64 bit integer (for using in 64 bit architectures).

Conventions

Graphic

Surely Snap7 VI icons will not be exposed to the New York Museum of Modern Art, but they follow a useful convention that helps to identify them at a glance.



Naming

As said, all the VIs access to the Snap7 through the LV interface library, so for each function we have three entity :

1. VI name
2. LV library function name
3. Snap7 function name

They are linked following this rule :

VI name	<object><Function name>
LV function	lv_<object>_<Function name>
Snap7 function	<object>_<Function name>

Example

VI name	SrvRegisterArea
LV function	lv_Srv_RegisterArea

Release

Everything you need is stored into LabVIEW folder that is divided as follow:

[\\Examples] contains a LabVIEW project which groups many examples. They are further divided into three folders (\\Client, \\Server and \\Partner) and are autonomous, i.e. you can run them without loading the project.

[\\lib] that contains the library Snap7.lvlib and all the interface vi.

[\\lib\\windows] contains lv_snap7.dll and snap7.dll, they are the deploy libraries.

[\\lib\\win32] contains 32 bit version of lv_snap7.dll and snap7.dll, they are the build libraries (see LabVIEW_32.bat).

[\\lib\\win64] contains 64 bit version of lv_snap7.dll and snap7.dll, they are the build libraries (see LabVIEW_64.bat).

[\\lib_build] contains three projects to compile lv_snap7.dll.

[\\lib_build\\VS2012_LV] Visual Studio 2012 solution.

[\\lib_build\\MinGW32] MinGW32 makefile and batch file for 32 bit.

[\\lib_build\\MinGW64] MinGW64 makefile and batch file for 64 bit.

[\\lib_src] contains the source files of lv_snap7.dll.

[\\lib_tmp] contains temporary compilation files (can be safely emptied).

LabVIEW 32 bit and LabVIEW 64 bit (native) use different library models, please follow these rules:

- If you plan to use Snap7 in 32 or 64 bit systems with 32 bit LabVIEW run **LabVIEW_32.bat** before opening any project.
- If you plan to use Snap7 in 64 bit systems with 64 bit LabVIEW run **LabVIEW_64.bat** before opening any project.

These batch files merely copy lv_snap7.dll and snap7.dll from the platform folder (win32 or win64) to the deploy folder (windows).

Final remarks

- Ø All the Snap7 blocks are **thread safe**.
- Ø All the Snap7 blocks, in which a string handle is passed, are memory safe, they check the string size against the Size parameter passed to avoid program crash. If the string size is less than the size param, the latter is trimmed, the function is performed but an error of partial data read or write is produced.
- Ø If you need to rebuild lv_snap7.dll and snap7.dll **use the same c++ compiler for both if you plan to use them in 64 bit environments**.
There is still no unified ABI convention for 64 bit systems, the problem is not the dll itself, but the .lib file needed to link them.
- Ø For lv_snap7.dll are valid all concepts exposed in "Rebuild Snap7".
- Ø **snap7.dll** must reside in the same folder of **lv_snap7.dll**, and their architecture must match (32/64 bit).

