FANUC DRIVER DEVELOPMENT KIT

This driver development kit enables you to create your own driver for Focas enabled Fanuc controllers within minutes with minimal effort. You can quickly implement any function from the Fanuc Focas Library and output data needed for your projects.

DESCRIPTION

The kit was built with Visual Studio 2017, you can use Visual Studio Community 2017 (free). It also was tested and can be used with other versions of Visual Studio >= 2008. Instructions can be found here. The kit contains 3 pre-built example functions from the Fanuc Focas Library. You can modify them or add your own items:

- Machine state information (uses cnc_statinfo)
- Parameter value (uses cnc_rdparam)
- PMC address value (uses pmc_rdpmcrng)

LICENSING

The driver development requires the most recent version of CNCnetPDM. It also works with a free license. However, in this mode you only get output for the first function in your DLL. With a valid license you are able to output the result of up to 30 functions per reading cycle, see licensing for details.

SETUP

- If you're using Visual Studio Community 2017 make sure that the following components are installed: Desktop Development with C++, Windows XP support for C++ and MFC and ATL support
- Extract all contents of fanucdeveloper.zip to a folder on your PC and navigate to this folder
- Copy customfanuc.dll from this folder to the directory where CNCnetPDM.exe is installed
- With Visual Studio 2017 simply open the project myfanuc.sln, for other versions of Visual Studio follow these instructions
- Compile the project once and make sure that you don't get errors.

USAGE

You only have to modify the content of file myfanuc.cpp to implement any function from the Fanuc Focas Library. Select Solution Explorer, open section Source Files and double click myfanuc.cpp.

INI FILE

Your device driver uses an own INI file that is automatically created by customfanuc.dll if it doesn’t exist. It enables you to activate/deactivate specific items and define Tag names for them. It always automatically gets the name customfanuc with the machine number added (as defined in CNCnetPDM.ini) e.g. customfanuc_1000.ini for device number 1000.

If it doesn’t exist customfanuc.dll creates it with 3 activated entries and one deactivated template that can be used for new functions.

The INI file contains numeric section identifiers. Each identifier contains information about an acquired item. For example

```
[2]
Active = 1
Name = AUTNR
Comment = Automatic mode number
Output section = 1
```

Means item number 2 should be acquired (Active = 1) and its Tag name is AUTNR. Tag names can have up to 5 characters. Output acquired by this function should be added to section 1, you can use up to 3 sections (1,2 or 3). Each section can contain up to 256 characters. For long items e.g. axis position it is recommended to use an own section.

Once the INI file is created its content can be manually modified at runtime without the need to restart the main service.

ADJUST ITEMS

Items can be adjusted and added in class DeviceReadExt. Most of the Fanuc Focas function calls use a specific structure that is mentioned in the documentation. For example cnc_statinfo uses structure ODBST. So, the first thing you have to do is to include this structure and assign a name for it e.g.

```
ODBST odbst; // Machine State structure
```

FIG 1: Include and name structure

Now you can call the function with

```
ret = cnc_statinfo(hFanuc, &odbst); // Machine status
if (!ret) // If it worked
       sprintf(m_stat.cReturn, "%d", odbst.aut);
```

FIG 2: Execute function

cnc_statinfo is the function, hFanuc is the handle and &odbst is a pointer to structure ODBST. If (!ret) means: Perform the following commands if the function does NOT return anything else than EW_OK (0).

If successful you can use the values returned by the function call to output them or perform additional calculations. The values are returned as structure members. ODBST is defined as

```
typedef struct odbst {
    short   hdck ;      /* handl retrace status */
    short   tmmode ;    /* T/M mode */
    short   aut ;       /* selected automatic mode */
    short   run ;       /* running status */
    short   motion ;    /* axis, dwell status */
    short   mstb ;      /* m, s, t, b status */
    short   emergency ; /* emergency stop status */
    short   alarm ;     /* alarm status */
    short   edit ;      /* editting status */
} ODBST ;
```

FIG 3: Members of structure ODBST

In this example these values can be directly used as odbst.aut or odbst.run a.s.o in your program.

The result of a successful function call is copied to mstat.cReturn which is automatically read by customfanuc.dll and added to the output section you defined.

Output sections contain one or more items in the form of tag name + pipe delimiter | + output value. You can add the result from multiple queries to a single section. 3 sections are available each one can contain up to 256 characters.

The sections are read by the main service which then creates database records or text files that contain GROUP_ID, READING_TIME, DEVICE_NUMBER, TAG_CODE, TAG_DATA, CREATION_DATE and PROCESSING_FLAG for each item acquired e.g.:

```
10000000001  2017-06-29 16:38:28.000     1001    AUTNR        1        2017-06-29
16:38:28.000   2
```

FIG 4: Output of records with tags and values

ADD ITEMS

It is highly recommended that you make a backup of file myfanuc.cpp (simply copy and paste it into the same folder) before adding items.

For an additional item it may be necessary to increase the number of commands in your INI file.

```
[GENERAL]
Commands = 6
```

FIG 5: Increase number of commands

To add the program number (modal O number) of the program which is currently selected at the controller proceed as follows.

Modify the template section [5] or add a new one e.g.

```
[6]
Active = 1
Name = MNPRG
Comment = Main Program
Output section = 1
```

FIG 6: Add an additional INI file section

Program number can be read by function cnc_rdprgnum which uses structure ODBPRO. In class DeviceReadExt add:

```
ODBPRO odbpro; // Program number
```

FIG 7: Assign a name for structure ODBPRO

Next, add an additional switch to class DeviceReadExt:

```
case 6:
ret = cnc_rdprgnum(hFanuc, &odbpro); // Program Number
if (!ret)
       sprintf(m_stat.cReturn, "%d", odbpro.mdata); // If successful add to output
break;
```

FIG 8: Call function for program number

You have successfully added program number to your driver!

Notes: As the program number is numeric you have to use %d as 2nd argument of the sprintf command. For text output you have to use %s. More information about these parameters can be found here.

DEBUG YOUR DRIVER

To debug your driver follow these steps:

Compile the debug version of myfanuc.dll.

Copy myfanuc.dll from the debug directory of your project to the folder where CNCnetPDM.exe is located. Make sure that customfanuc.dll and all Fanuc Focas DLL files are also in this directory. Install one of the Fanuc device drivers if not.

Change the content of section [RS232] of CNCnetPDM.ini to use customfanuc.dll e.g.:

```
1 = 1001;19200;8;N;1;DEV #1;192.168.1.100;8193;0;FANUC;7;6711;none;none;0;customfanuc.dll
```

Start CNCnetPDM and make sure it uses your new driver. You can see that because CNCnetPDM copies customfanuc.dll and adds the machine number to it e.g. customfanuc_1001.dll. If no INI file for machine 1001 exists it also creates a new INI file with 3(!) active entries e.g. customfanuc_1001.ini.

Then, check the content of the log file for your device in subfolder \log (File name = log_ + machine number + _ + date + .txt). Errors reported by the device driver are added to this file. For example 'Error(s) reported by device 1000: INIT: Windows socket error' means that your controller is not reachable at all.

Note: If you have added new items or changed existing ones please adjust the INI file (number of items, sections) for the device according to your definitions in myfanuc.cpp before starting CNCnetPDM.

If your new items are missing or incorrect, stop the service, fix the error, rebuild myfanuc.dll, copy it again to the folder with CNCnetPDM.exe and adjust your INI file. Then, start CNCnetPDM.

To debug the driver set at least one breakpoint in myfanuc.cpp. A good place to start is in class DeviceReadExt in the line that starts with: switch (iPass)

In Visual Studio (with CNCnetPDM running!) click Debug -> Attach to Process, select CNCnetPDM.exe and click Attach.

As soon as the breakpoint is reached you can follow the execution of the program by clicking F10 and see the results of function calls and values of variables.

Note: If you make changes to your source code and rebuild the driver please adjust the INI file for the device (e.g. customfanuc_1001.ini) before starting to debug again.