VECTOR

How to Use C-Code Functions in CANape

Version 2.1
2024-07-24
Application Note AN-IMC-1-012_How_to_use_C-Code_Functions_in_CANape.docx

Author Vector Informatik GmbH, PMC92
Restrictions Public Document
Abstract This document describes how to use C-code functions in CANape.

Table of Contents

1.0 Overview

In CANape, it is possible to use the installed programming language CASL to develop functions and scripts, e.g. in analyzing measurement data or automating frequently used sequences. This document describes how the supplied functional library can be extended to include your own libraries or functions. Potential applications include functions whose source code should not be published, or algorithms that cannot be implemented with the existing library or only with great effort.

In the directory `C:\Users\Public\Documents\Vector\CANape Examples <version number>` of the CANape installation, you can find an example. It describes how to use your own functions including source code and how to integrate it in CANape.

1.1 Files From the Sample Project

The sample project `FunctionDll` is automatically installed with the CANape installation. Additionally, in CANape Tools & Demos you can find more information about the sample project.

1.2 CANape Files

The following files you can find in the directory ...`\CANape Examples <version number>\FunctionDemoDll` of your CANape installation:

`CANape.INI`	CANape project file
`DemoDll.cna`	CANape configuration file
`demodll.dll`	Sample file with auxiliary functions
Subfolder `Panels`	Several CANape panels for starting a sample script, e.g. `Panel1.xvp`, `DemoDll.cnp`
Subfolder `Scripts`	Several sample scripts, e.g. `StartProject.cns`, `StartMeasurement.cns`, `CloseXCPSimulator.scr`

1.3 Demo DLL Files

The following demo DLL files you can find in the directory …`\CANape Examples <version number>\FunctionDemoDll\Sources` of your CANape installation:

`functiondll.h`	Header file with auxiliary functions of the DLL
`demodll.cpp`	Source file with sample implementation
`demodll.dsp`	Microsoft Developer Studio Visual C++ 6.0 project file
`demodll.dsw`	Microsoft Developer Studio Visual C++ 6.0 workspace

2.0 Sample Code

In order to call your own functions in CANape, they must be defined, declared and implemented in the sample code. It is easy to modify the `Demodll.cpp` file for this purpose.

2.1 Listing of Functions

In the `sDllInterfaceDescriptorTable` table, functions are listed according to the following scheme:
{ "Function name", "VALUE/REFERENCE Argument" },

Arguments are of the type VALUE or REFERENCE. In the case of VALUE, a copy of the parameter value is passed to the function (input parameter). Changes made to the parameter value in the function remain void for the calling program (call by value). In the case of output parameters – if the parameter value should be changed by the function – please use the type REFERENCE (call by reference). Arrays and strings are essentially passed by REFERENCE.

In all cases, the return value of the function is of the type double.

Both, the function name and the names of the formal parameters (arguments) may be user-defined. The function name must agree with the implementation.

Note
Parameter names are shown on the status bar in CANape's Function Editor, when you select the related function with the cursor in the Function Editor's shortcut menu. Therefore, please use descriptive parameter names.

Each function may contain as many arguments as desired. They just need to be separated by commas.

Make sure that the end of the table is properly terminated!

```
//---------------------------------------------------------------------------

// Define the exported functions here

//---------------------------------------------------------------------------

// Allowable parameter types:

//  VALUE  Single value "Call by Value"

//  REFERENCE Single value or array "Call by Reference"

//---------------------------------------------------------------------------

const static VDllInterfaceTable sDllInterfaceDescriptorTable[]=

{

  { "ArithMinV", "VALUE arg1, VALUE arg2" },

  { "ArithMaxV", "VALUE arg1, VALUE arg2" },

  { "ArithMinB", "REFERENCE arg" },

  { "ArithMaxB", "REFERENCE arg" },

  { "ArithMiddleB", "REFERENCE arg" },

  { "ChangeBuffer1D", "REFERENCE arg1, REFERENCE arg2" },
```

```
  { "ChangeBuffer2D", "REFERENCE arg1, REFERENCE arg2" },

  { "RotateString", "REFERENCE arg1, REFERENCE arg2" },

  { NULL, NULL } // MUST: table termination

};
```

2.2 Declaring and Implementing Functions

The following scheme is used for both declaration and implementation:

```
// Declare Exported Functions

MYAPI(double) ArithMinV(VDllFunArgVariant * arg, int argCnt);
```

Only the function name (`ArithMinV`) needs to be modified here. The variable `argCnt` contains the number of formal parameters of the function. This agrees with the number of parameters shown in the listing of functions in the table `sDllInterfaceDescriptorTable` (see section 2.1).

When calling the function, `arg` contains the list of parameter values, wherein the individual data types apply to the passed parameters.

```
//---------------------------------------------------------------------------
----

// ArithMinV

//---------------------------------------------------------------------------
----

// Example function to show how to work with values

//---------------------------------------------------------------------------
----

// Definition: { "ArithMinV", "VALUE arg1, VALUE arg2" },

//---------------------------------------------------------------------------
----

// Script:

// double arg1, arg2, minimum;

// arg1= 3.14;

// arg2= 3.1415;

// minimum= ArithMinV(arg1, arg2);

// write("Minimum from %f and %f is %f", arg1, arg2, minimum);

//---------------------------------------------------------------------------
----

MYAPI(double) ArithMinV(VDllFunArgVariant * arg, int argCnt)

{

  double arg1= DBL_MAX, arg2= DBL_MAX;
```

```
  // Check arguments
  if (argCnt != 2)  return DBL_MAX;
  if (arg[0].type == kDclassUnknown) return DBL_MAX;
  if (arg[0].type == kDclassString)  return DBL_MAX;
  if (arg[1].type == kDclassUnknown) return DBL_MAX;
  if (arg[1].type == kDclassString)  return DBL_MAX;

  // Get doubles
  if (arg[0].type == kDclassFloat) arg1= (double) arg[0].data->f;
  if (arg[0].type == kDclassInt) arg1= (double) arg[0].data->l;
  if (arg[0].type == kDclassUint)  arg1= (double) arg[0].data->ul;
  if (arg[0].type == kDclassDouble)  arg1=  arg[0].data->d;
  if (arg[1].type == kDclassFloat) arg2= (double) arg[1].data->f;
  if (arg[1].type == kDclassInt) arg2= (double) arg[1].data->l;
  if (arg[1].type == kDclassUint)  arg2= (double) arg[1].data->ul;
  if (arg[1].type == kDclassDouble)  arg2=  arg[1].data->d;

  // Return the minimum
  return (arg1 < arg2) ? arg1 : arg2;
}
```

2.2.1 Supported Objects and Data Types

The return values of functions of an external function library (DLL) are all type `double`.

Any of the simple data types of CANape's programming language may be used as parameters of a function:

- 8 bit signed/unsigned integer
- 16 bit signed/unsigned integer
- 32 bit signed/unsigned integer
- float (32 bit)
- double (64 bit)

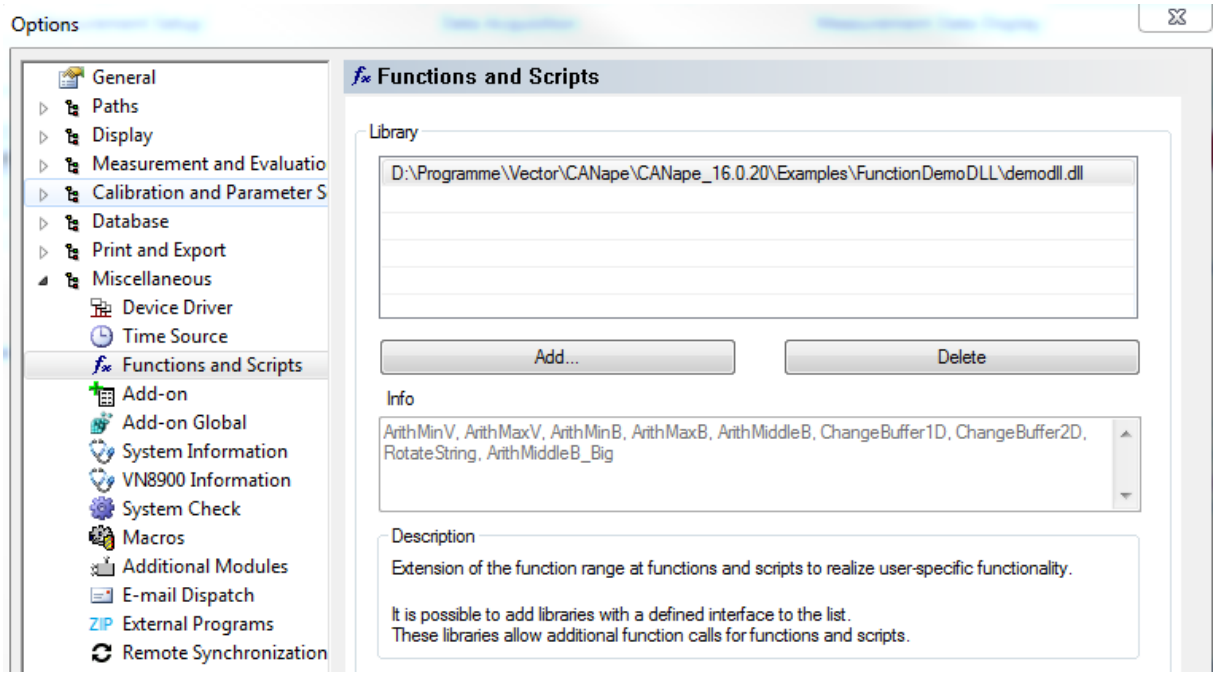
Furthermore, strings as well as one- and two-dimensional arrays of the simple data types may be passed. Characteristic curves and maps may also be passed to functions. Please note that only function values of the characteristic curve/map may be passed (not the X/Y axis points). When a function is called with a measurement variable as a function parameter, the current or last signal value is always passed (for arrays, the array of values is passed).

It is not possible to use 64-bit integer data types in scripts and functions.

3.0 Integration of Functions in CANape

The DLL is integrated in CANape by using the options dialog:

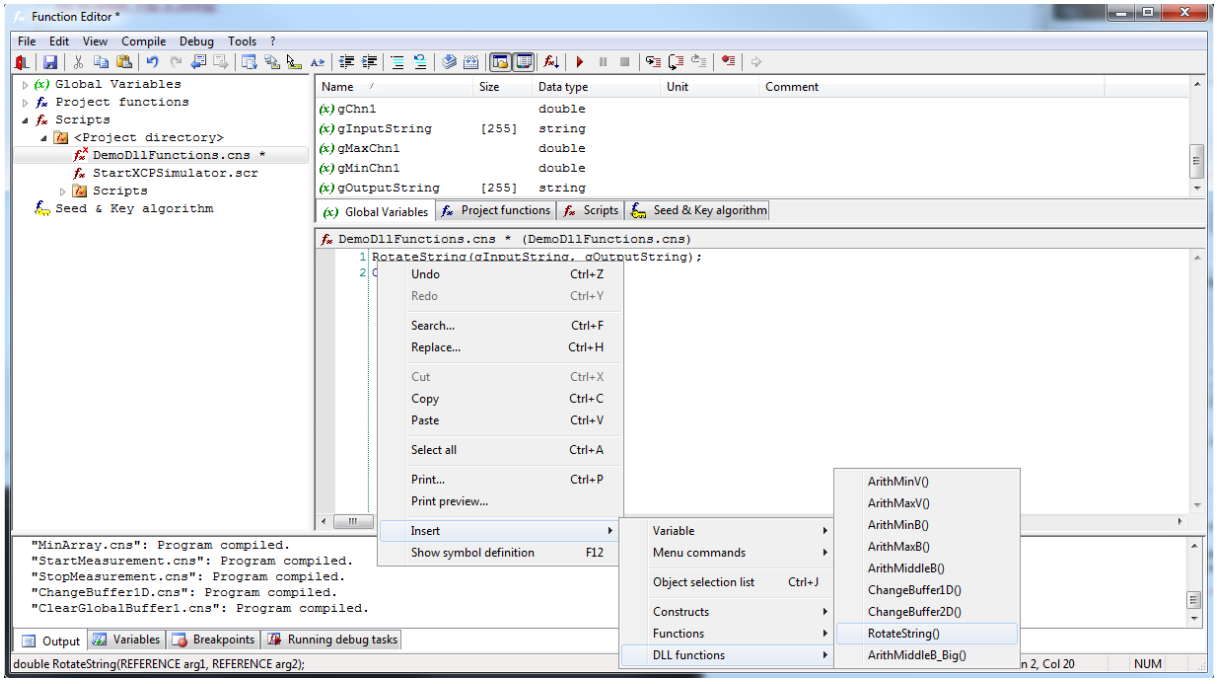
1. Open the **Options** dialog in inside the **Backstage** area**.**
2. Open **Miscellaneous|Functions and Scripts** page.
3. Use the button **[Add]** to select the appropriate DLL from the file explorer.



4.0 Use in Functions Editor

After the DLL has been integrated, you can call the functions in the CANape Functions Editor.

When you write a function or a script for example, you can access the functions in your DLL via the context menu (press right mouse key). The path is **Insert|DLL Functions|RotateString()** for example.



Note

For a more detailed description see our manual `CANape CASL - Calculation and Scripting Language` in the Vector download center on www.vector.com.Here you can obtain supplemental information.

5.0 Contacts

For a full list with all Vector locations and addresses worldwide, please visit http://vector.com/contact/.