ADVANCED MOTION CONTROLS

Using the SMEM to Interact with Click&Move®

Introduction

The shared memory (SMEM) interface is a way inside of Click&Move® to share data between multiple processes running on the same machine. The pre-generated library ensures data consistency so that is not the worry of the developer to allow easy read and write access to Click&Move®.

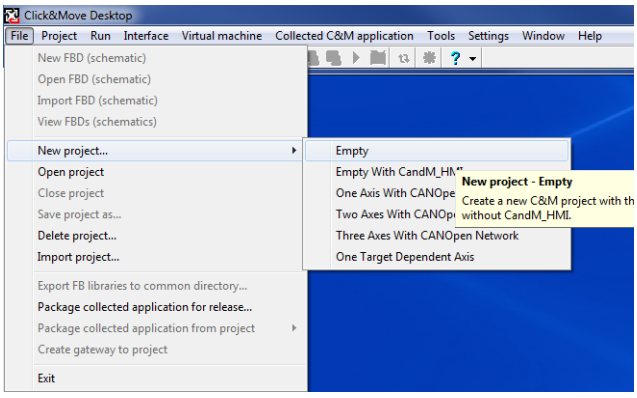
This procedure will review the process of setting up an SMEM interface and interacting with it via C++ by demonstrating an example to calculate the hypotenuse of a right triangle.

Pre-Requisites

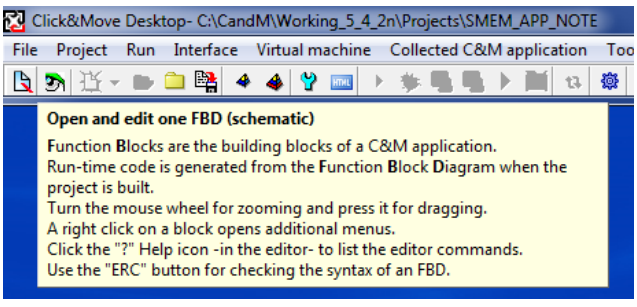
- Click&Move® IDE installed on your local machine
- Microsoft Visual Studio Community

Setting up Click&Move®

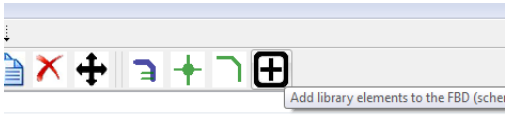
1. Open Click&Move and create a new empty project by selecting *File > New Project > Empty*.

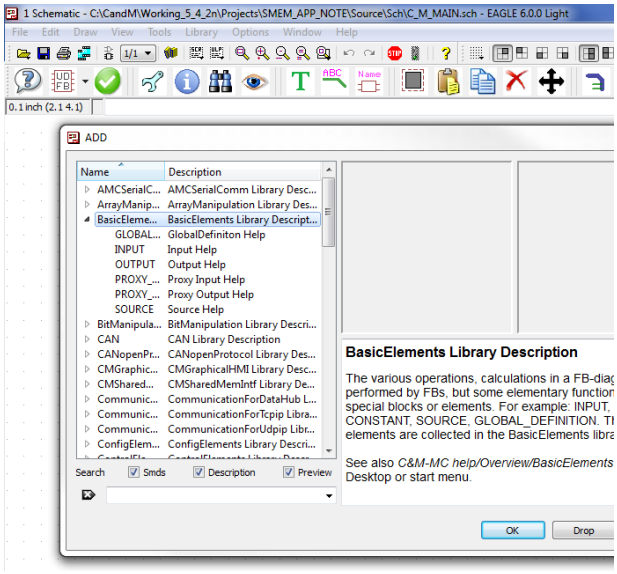


2. Click the **Open Schematic** icon, and select the *C_M_MAIN.sch* schematic.

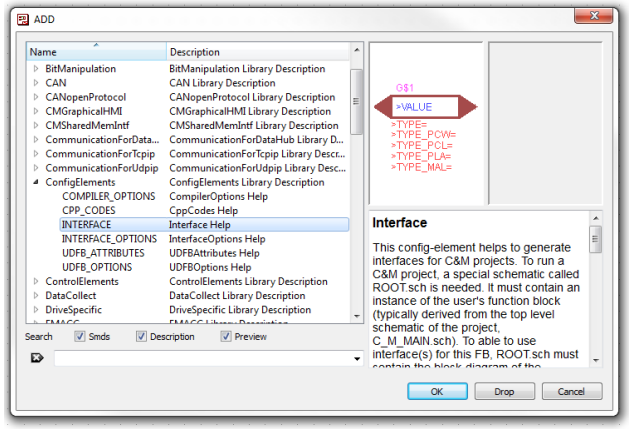


3. Click the **Add Library Element** icon, and add 3 INPUT and 3 OUTPUT elements to the main schematic.

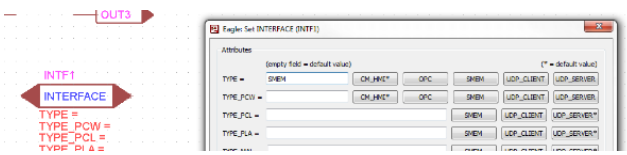




4. Also add an INTERFACE block to set up the SMEM.

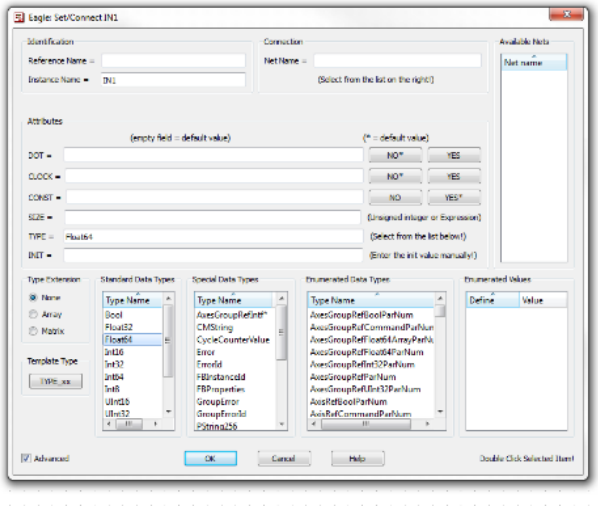


5. To set the interface type, right-click the Interface Block and set the type to SMEM.

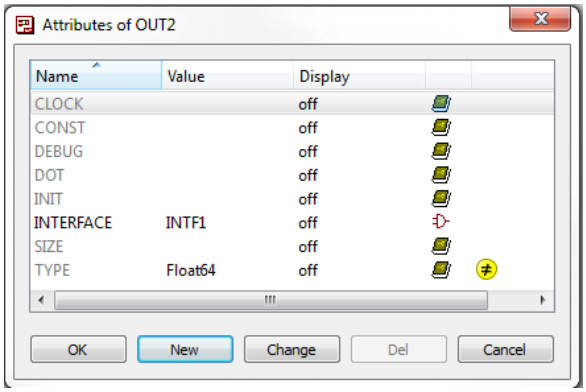


6. Set the Reference Names of the INPUT and OUTPUT elements as given in the table below by right clicking each element and clicking *C&M Set/Connect*. The data type of each block also needs to be set to Float64 by clicking the *Advanced* checkbox in the bottom left of the element window, and by setting the Standard Data Type to Float64. OUT1, OUT2, and IN3 will be named LEG_A_SMEM, LEG_B_SMEM, HYP_C_SMEM, and also have a datatype of Float64.

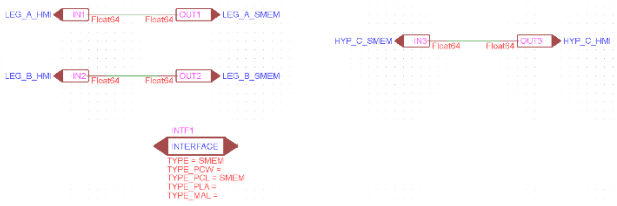
Element	Name
IN1	LEG_A_HMI
IN2	LEG_B_HMI
IN3	HYP_C_SMEM
OUT1	LEG_A_SMEM
OUT2	LEG_B_SMEM
OUT3	HYP_C



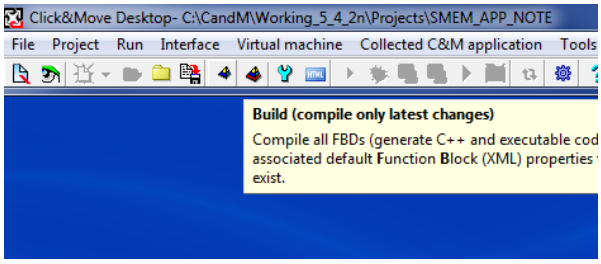
7. Now add a custom attribute by right clicking each INPUT/OUTPUT function block and then selecting *Attribute*. Inside of the Attribute dialog box add a *New* attribute named INTERFACE, and a value of INTF1 for OUT1, OUT2, and IN3. The rest will just default to the Mini-HMI.



8. Finally, connect all your SMEM OUTPUT blocks to your Mini-HMI INPUT blocks, and your Mini-HMI OUTPUT blocks to your SMEM INPUT blocks. Your diagram should at this point look like the following:

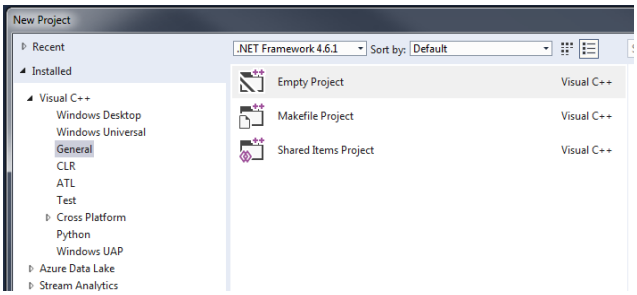


9. Once these are named properly and the data types are set, the project is ready to be built. Save your schematic, exit out of the schematic builder and click the **Build** icon on the main Click&Move menu bar.

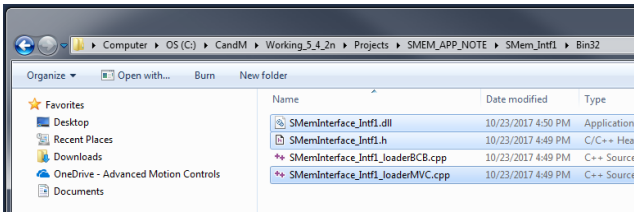


Setting up Visual Studio and Loading the Libraries

1. Once the project is built, open a new *General* Visual C++ project in Visual Studio.

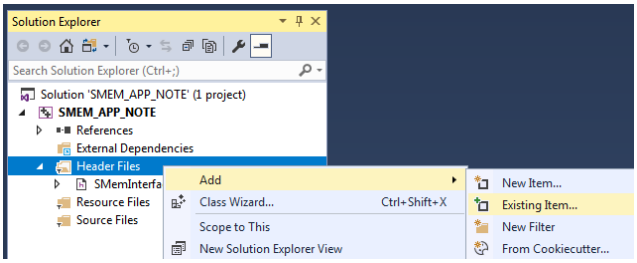


2. Under your project folder, there are 3 libraries and 2 C++ files needed to interact with the SMEM interface. The main library as well as the header and main C++ file are located in **\Project_Folder_Location\SMem_Intf1\Bin32**. Copy SMemInterface_Intf1.dll, SMemInterface_Intf1.h, and SMemInterface_Intf1_loaderMVC.cpp to your MS Visual Studio Project Folder.



3. The other two libraries that you will need to copy to your Visual Studio Working directory are libgcc_s_dw2-1.dll and libstdc++-6.dll. These files are located in **ClickAndMoveHome\Examples\DLL_MotionController_X3\Test_program**.

4. Now load the C++ header file into Visual Studio by right-clicking the header file and clicking *Add->Existing Item*, and add SMemInterface_Intf1.h. Then in the same fashion load SMemInterface_Intf1_loaderMVC.cpp into your source file folder.



5. Before the getter and setter methods can be used to interact with the shared memory interface, the shared memory name variable will need to be declared in your main function. Add the following line to use the default name for the shared memory interface:

```
char SHARED_MEMORY_NAME[] = "DefaultSmemInterface";
```

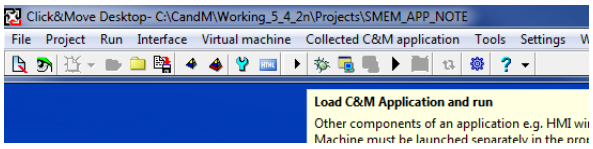
6. Inside of the pre-built code there is a comment saying the shared memory is opened successfully. After this line you can begin interacting with the shared memory interface.

   The following code will be used to interact with the SMEM and calculate the hypotenuse of a right triangle:

```
// Shared memory has been opened successfully.
  // Your application can use the global function
pointers from this point...

  double leg_a;
  double leg_b;
  double hyp_c;

  // Run process until it is killed
  while (true) {

            // Get values from memory and assign
to variable
            GetLegASmem(smemRef, (::UInt8*)&leg_a,
100);
            GetLegBSmem(smemRef, (::UInt8*)&leg_b,
100);

            // Calculate the hypotenuse using
functions from the math.h package
            hyp_c = sqrt(pow(leg_a, 2.0) +
pow(leg_b, 2.0));

            // Set value of hypotenuse in the SMEM
            SetHypCSmem(smemRef, (::UInt8*)&hyp_c,
100);

            // Sleep for 1 millisecond to not over
tax the processor
            Sleep(1);
  }
```

Running the Program

1. Once that code is in place and builds and runs in Visual Studio, you can keep this program running and also load and run your Click&Move program.



2. Once it is running you should be able to type in leg lengths, and the C++ program will calculate the hypotenuse and return it to Click&Move.