

Obtaining Interactive Control of an MCU and Development Tools using HTS

Description: The purpose of this lab shall be to familiarize the user with HTS.

Objectives

1. Provide an understanding of how to install / uninstall HTS.
2. Give a basic example of HTS functionality.
3. Walk the user through a C++ example of coding HTS in Windows.

Lab Materials:

Please verify you have the following materials at your lab station.

- HEW with HTS
- Target Renesas MCU
- Microsoft Visual Studio C++ Express 2008

Skill Level : Intermediate. User must have basic knowledge of Renesas tools and experience with C++ programming in Windows.

Time to Complete Lab: 60 minutes

Lab Sections

- | | |
|---|--|
| <p>1 Setting up HTS</p> | <p>Time to complete task: 10 minutes</p> |
| <p>2 Getting familiar with HTS</p> | <p>Time to complete task: 10 minutes</p> |
| <p>3 Windows programming with HTS – beginning</p> | <p>Time to complete task: 40 minutes</p> |
| <p>4 Windows programming with HTS – advanced</p> | <p>Time to complete task: 40 minutes</p> |

1

Setting up HTS

Time to complete task: 10 minutes

Overview:

While HEW Target Server is included with HEW, it is not part of the normal install process. Before using HTS, it must be registered both with Windows and HEW, which require two separate processes. There is also a complication in that once registered, updates of HTS which are obtained via the HEW Auto Updater program are automatically registered with HEW, but still require a manual update of their registration within the Windows environment. To do so requires the user to have knowledge of how to unregister HTS from Windows. We'll start by unregistering HTS from Windows and HEW, then registering them to ensure the user is familiar with both procedures.

Procedural Steps



For the sake of simplicity, we will assume that the Renesas HEW tools are installed at **C:\Program Files\Renesas\Hew**

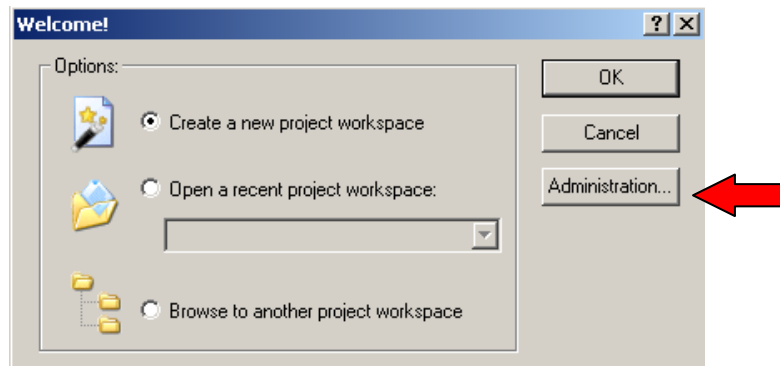
1. On the desktop, go to **My Computer>>C Drive>>Program Files>>Renesas>>Hew**
2. Locate and run the file:

```
UNREGISTERSERVER.BAT
```

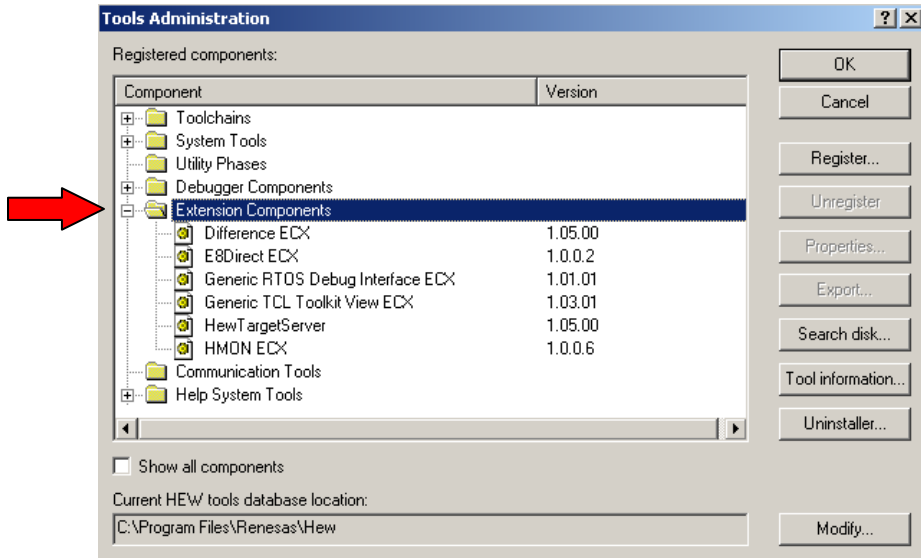


Running UNREGISTERSERVER.BAT causes HTS to be removed from the Windows Registry.

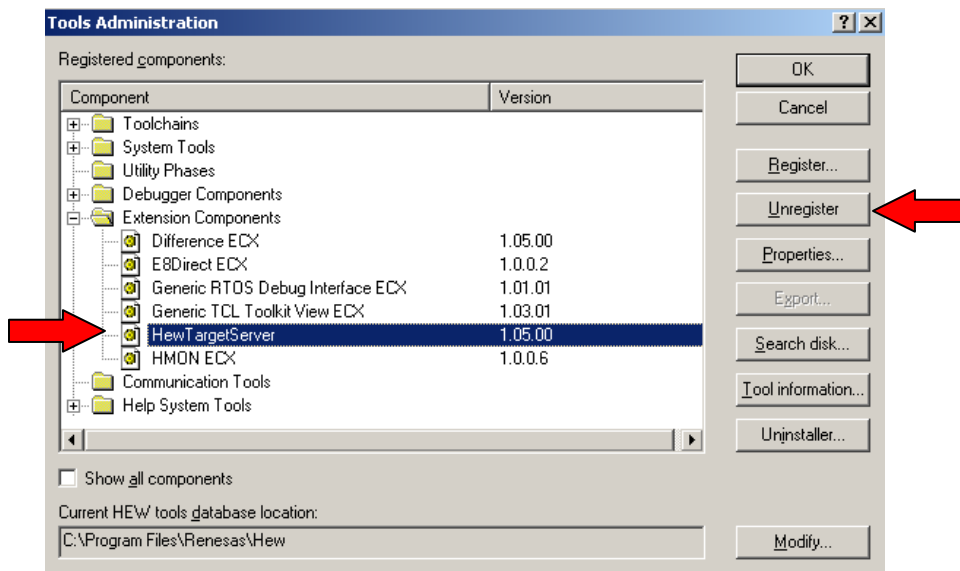
3. Close the window.
4. Start **HEW** by double clicking on the HEW icon on the desktop
5. When HEW opens and displays the *Welcome!* dialog box, click on the **Administration** button.



6. When the *Tools Administration* dialog box appears, expand the **Extension Components** tree.

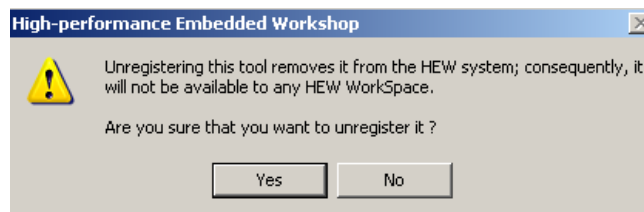


7. Click once to select the **HEWTargetServer** listing.



8. Click the **Unregister** button.

9. A warning box will appear indicating you are about to unregister a tool. Click **Yes**.



10. Click **OK** to close the *Tools Administration* dialog box.

11. Click **Cancel** on the *Welcome!* dialog box.

12. Exit HEW.



At this point, we have successfully unregistered HTS from both Windows and HEW. It is necessary to know this procedure when updating HTS versions as you must manually unregister old version first. Next we will follow the steps to register HTS in both Windows and HEW. As expected, the steps are very similar.

13. On the desktop, go to **My Computer>>C Drive>>Program Files>>Renesas>>Hew**

14. Locate and run the file:

REGISTERSERVER.BAT



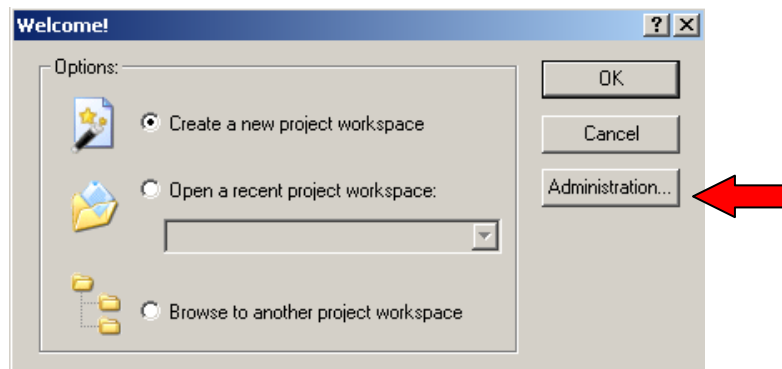
Running REGISTERSERVER.BAT causes HTS to be registered into the Windows Registry.

15. Close the window.

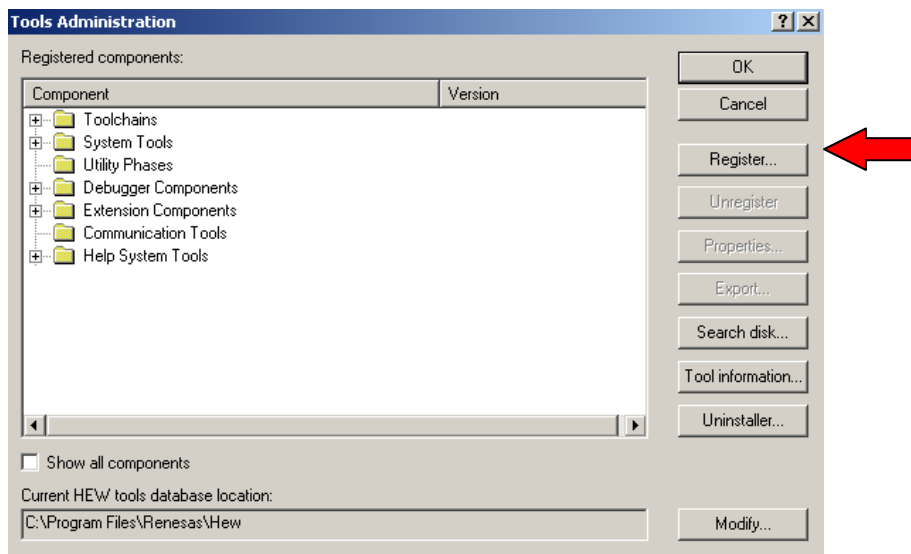


16. Start HEW by double clicking on the HEW icon on the desktop

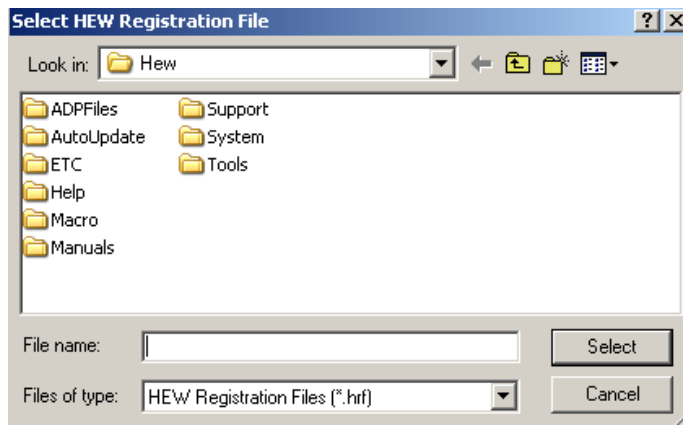
17. When HEW opens and displays the *Welcome!* dialog box, click on the **Administration** button.



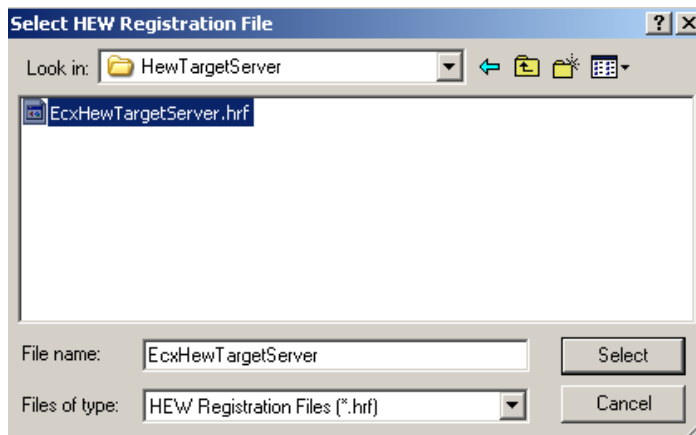
18. When the *Tools Administration* dialog box appears, click the **Register** button.



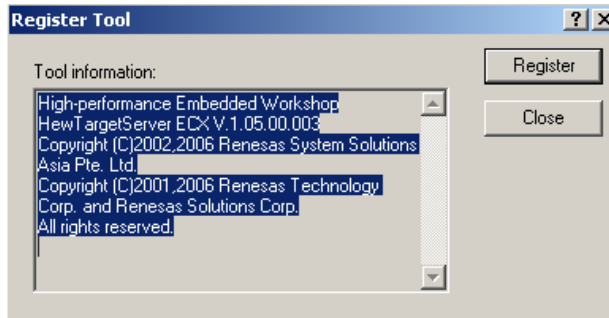
19. In the *Select HEW Registration File* dialog box that appears, go to **System>>Sec>>HewTargetServer**.



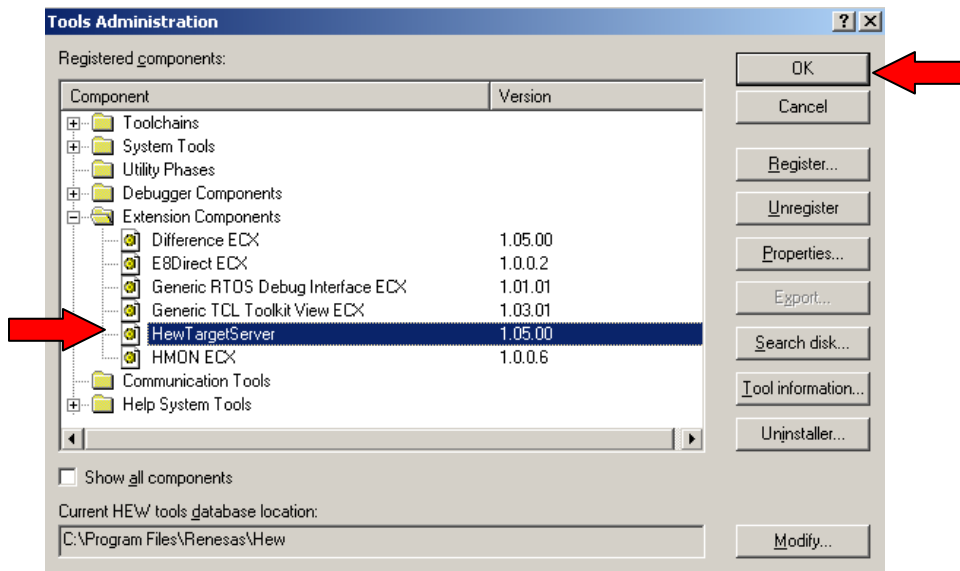
20. Choose the **EcxEWTargetServer.hfr** file and click **Select**.



21. The *Register Tool* dialog box will appear with version info on HewTargetServer. Click the **Register** button.



22. HewTargetServer and its version will now appear under the *Extension Components* tree. Click **OK** to close the *Tools Administration* dialog box.



23. Click **Cancel** on the *Welcome!* dialog box.

24. Exit HEW.



We have now successfully unregistered and registered HewTargetServer from both Windows and HEW. As previously mentioned, it is important to know both procedures when working with and updating HTS.

This is the end of Section 1.

2

Getting familiar with HTS

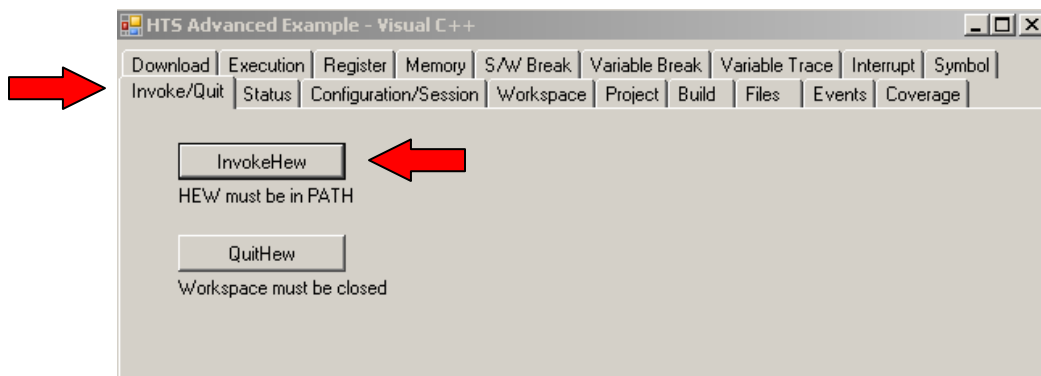
Time to complete task: 10 minutes

Overview:

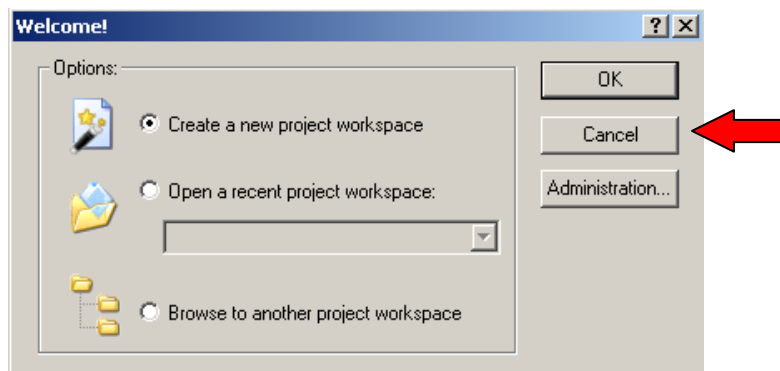
This section of the lab will provide a basic overview of some of the power and features of HewTargetServer. In this example we will use the HTS Advanced Example program to start HEW, load a project workspace, build the program, download the program, start and stop the programming running on the target MCU, and verify status messages from HEW. All of this shall be accomplished using HewTargetServer and the HTS Advanced Example program with very minimal manual interaction with HEW itself.

Procedural Steps:

1. Launch the **HTS Advanced Example** program by double clicking on it's icon on the Desktop.
2. Move the *HTS Advanced Example* into a corner of screen so you can see and work with it, but it is not in the center of the screen.
3. Connect the **HTS board** to the computer.
4. On the *Invoke/Quit tab*, click the **InvokeHEW** button to tell HTS to start HEW.



5. Once HEW has started, jump over to the HEW program, click **Cancel** on the *Welcome!* dialog box.

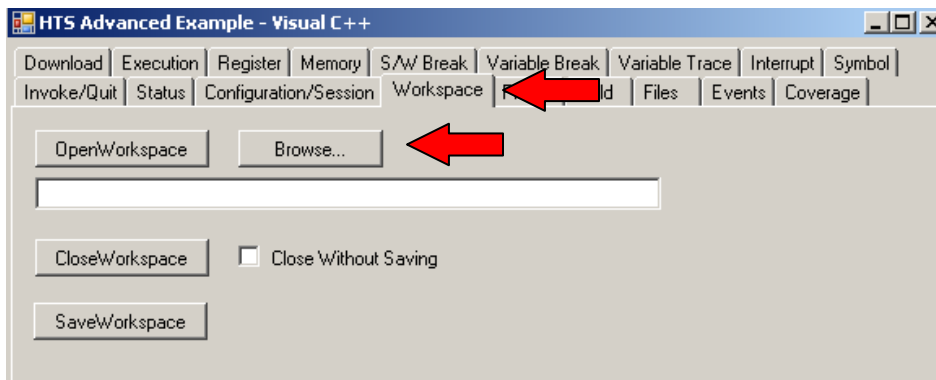


6. Jump back to the *HTS Advanced Example* program.



Now we have HEW running. Next we need to open a project workspace to use.

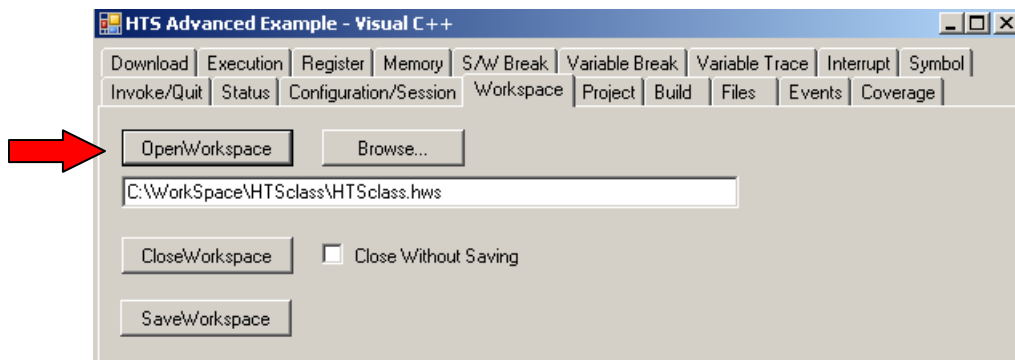
- On the *Workspace* tab, click the **Browse** button.



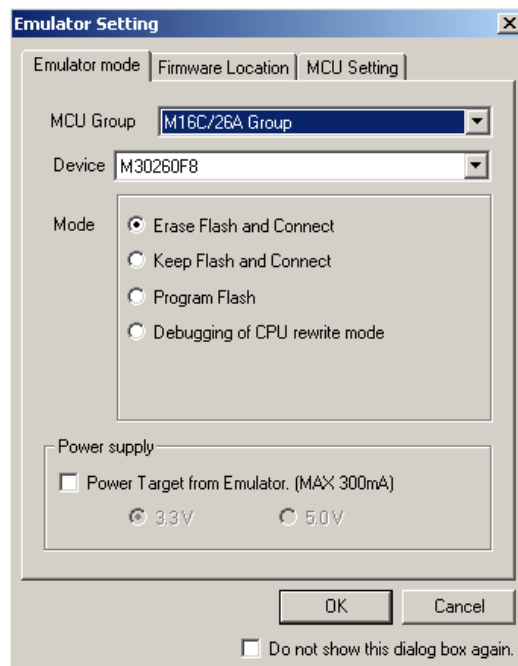
- In the *Open* dialog box that appears, browse to **C:\Workspace\HTSclass\Lab2** and select the file:

Lab2.hws

- Click the **OpenWorkspace** button to tell HEW to open the workspace we have selected.

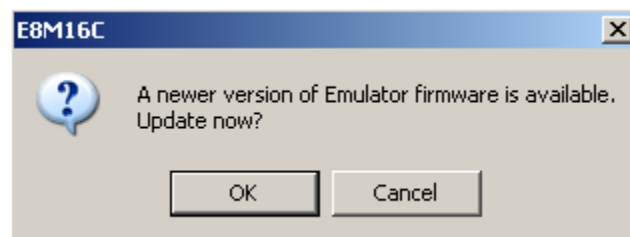


- Jump over to *HEW* and verify a workspace has opened. You will see a list of files on the left side of the program and the Emulator Setting dialog box shown below.



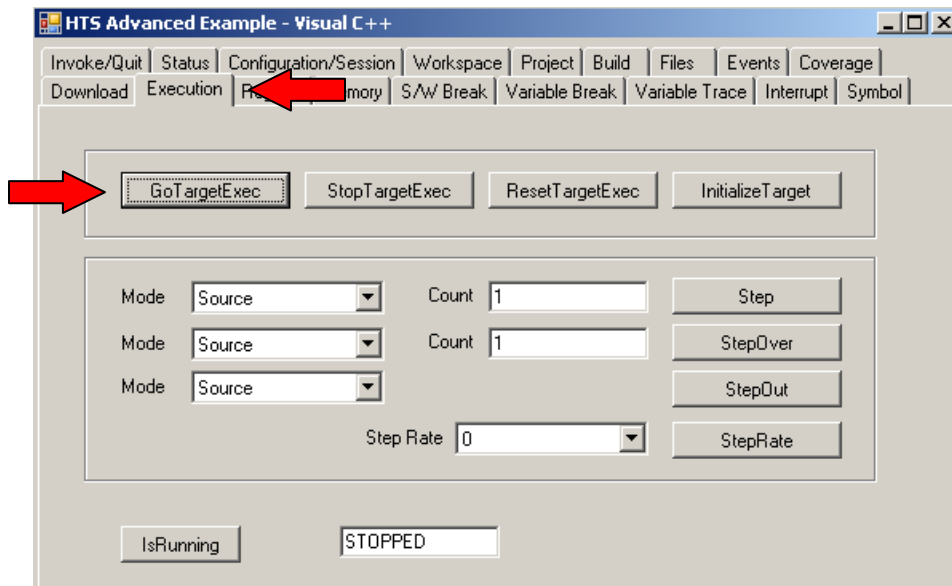
11. Click **OK** on the *Emulator Setting* dialog box.

12. Depending on the date which your *HTS board* was manufactured, you may see the following message asking to update your Emulator firmware. If this is the case, click **OK** to allow the automatic update.

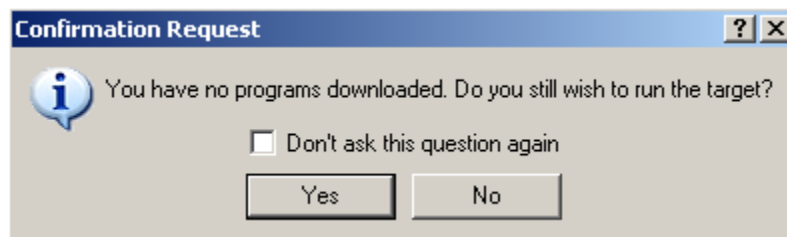


13. In the *HTS Advanced Example*, click on the **Execution** tab.

14. Click the **GoTargetExec** button.



15. In *HEW*, verify the dialog box below has appeared. Click **No**.

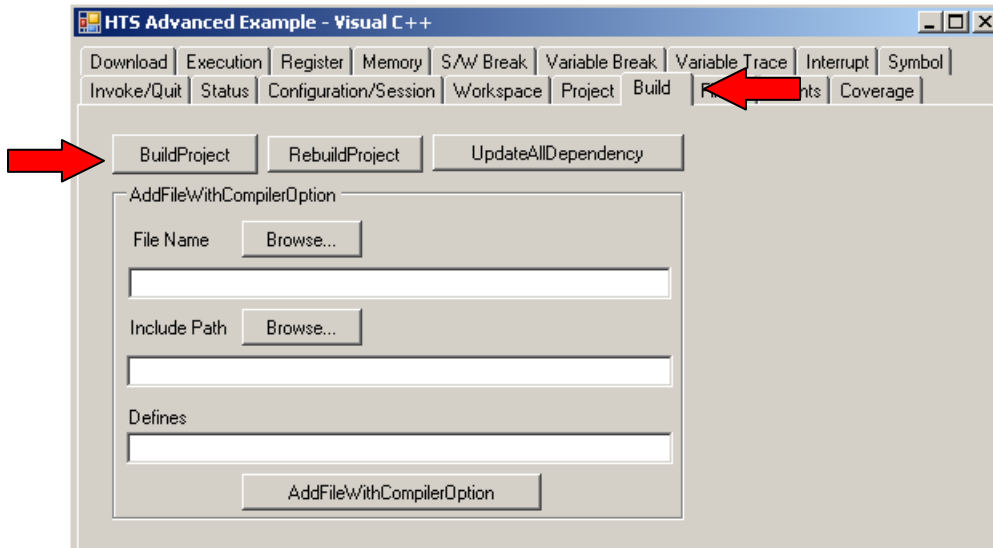


Question: (Write Answers on Sheet at the End of the Lab)

2.1) Why did the *Confirmation Request* message appear?

16. In the *HTS Advanced Example*, click on the **Build** tab.

17. Click the **BuildProject** button to build the project.



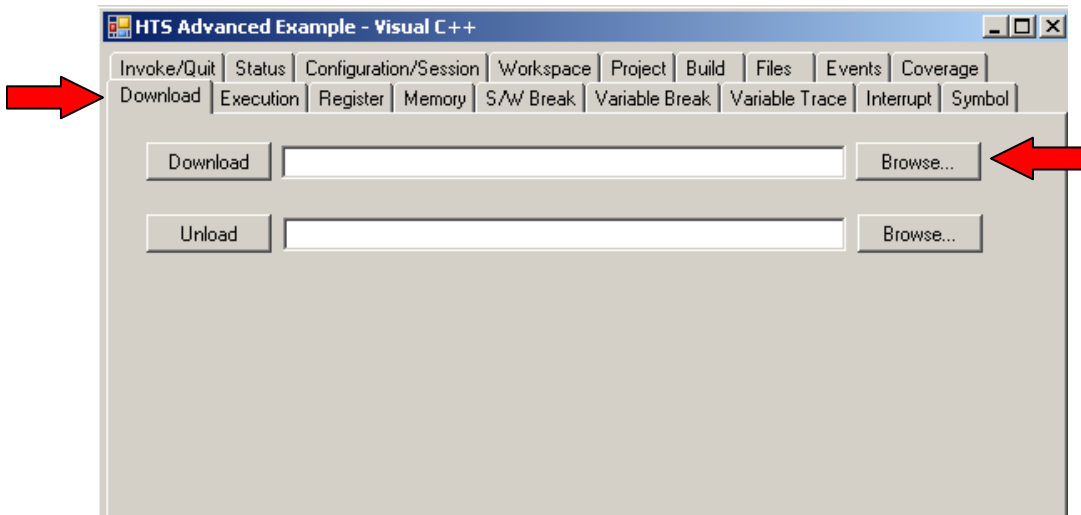
18. In *HEW*, verify the project was built by observing the following text in the **Build** window at the bottom of *HEW*:

```
Build Finished
0 Errors, 0 Warnings
```

19. A dialog box will also appear in *HEW* asking "OK to download module:xx". Click **Cancel**.

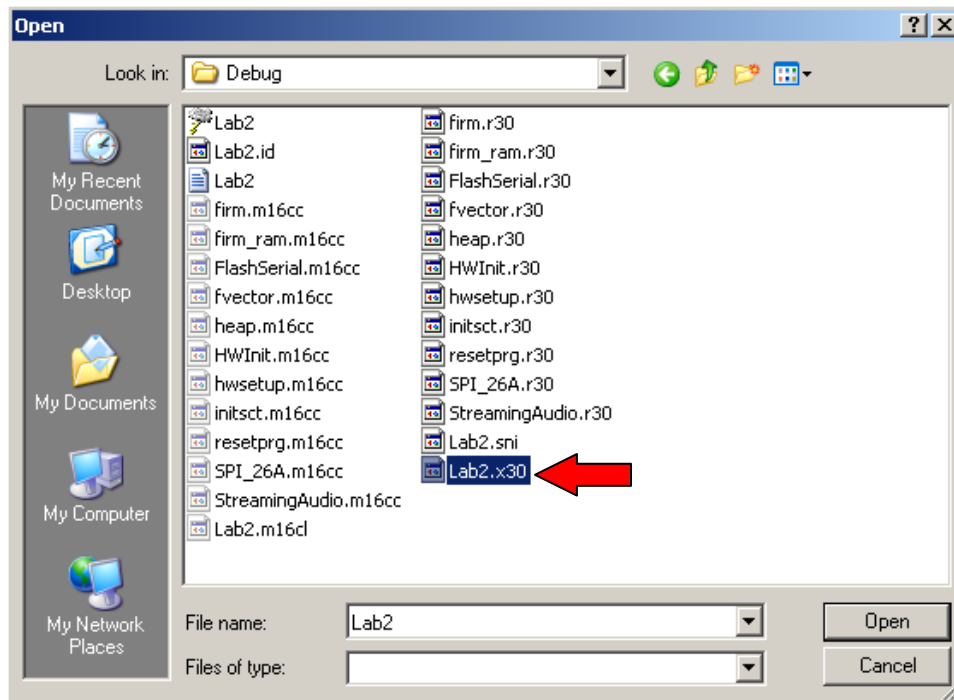
20. In the *HTS Advanced Example*, click on the **Download** tab.

21. Click on the **Browse** button for the *Download* instruction.

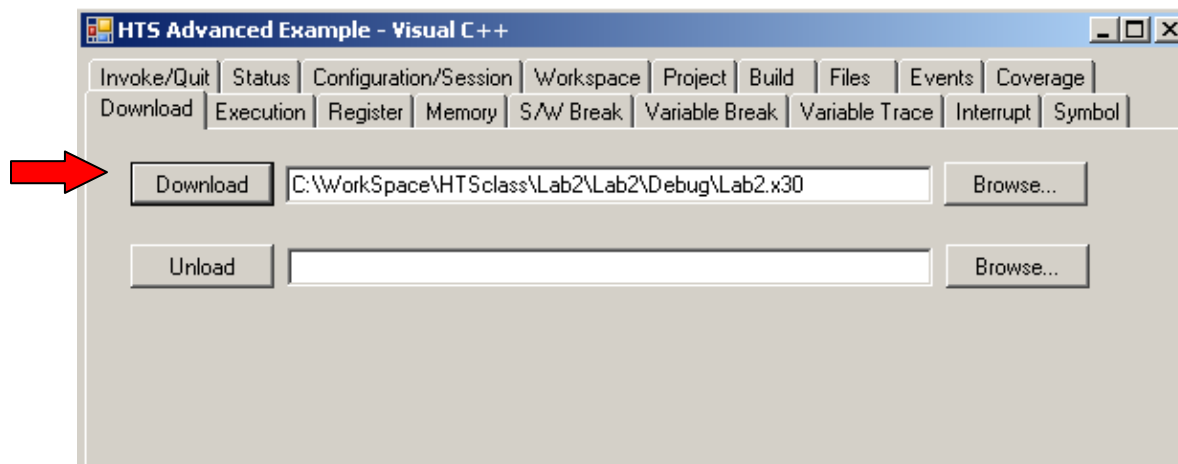


22. In the *Open* dialog box that appears, browse to **C:\Workspace\HTSclass\Lab2\Lab2\Debug** and select the file:

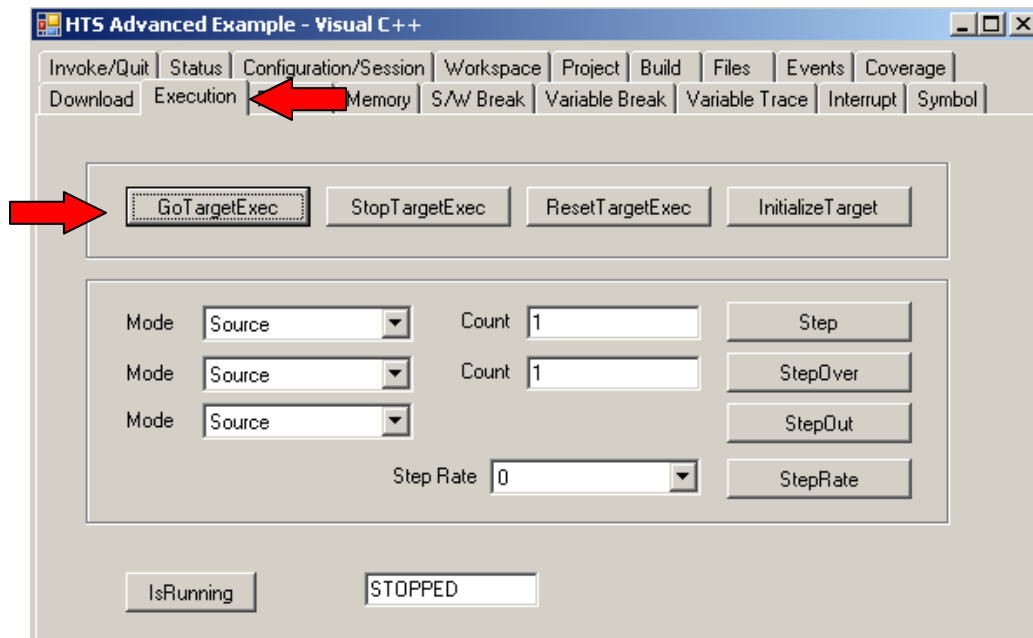
Lab2.x30



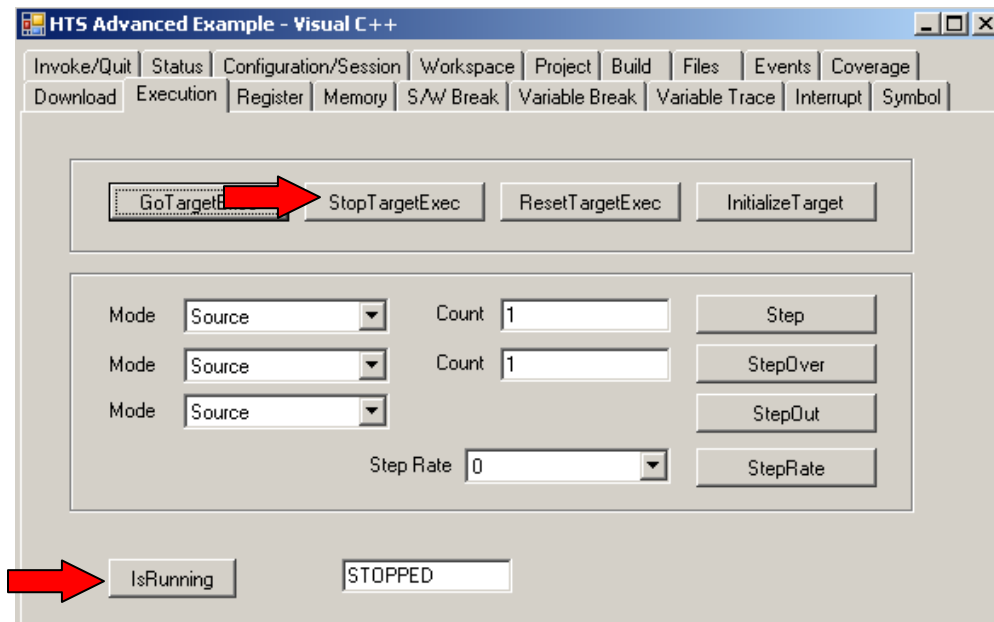
23. Click on the Download button to issue an HTS command for HEW to download the code to the HTS board.



24. On the Execution tab click the **GoTargetExec** button.



25. On the *HTS board*, verify the **LEDs** are blinking indicating the program is now running.
26. In the *HTS Advanced Example*, click the **IsRunning** button.
27. Click the **StopTargetExec** button.
28. Click the **IsRunning** button again.



29. In *HEW*, observe the MCU's Program Counter (yellow bar highlight code)
30. On the *HTS board*, verify the LEDs have stopped blinking.

Question: (Write Answers on Sheet at the End of the Lab)

2.2) What is the purpose of the IsRunning button?

31. Click the **ResetTargetExec** button.



32. In *HEW*, once again observe the MCU's Program Counter (yellow bar highlight code)

33. Click the **GoTargetExec** button again.

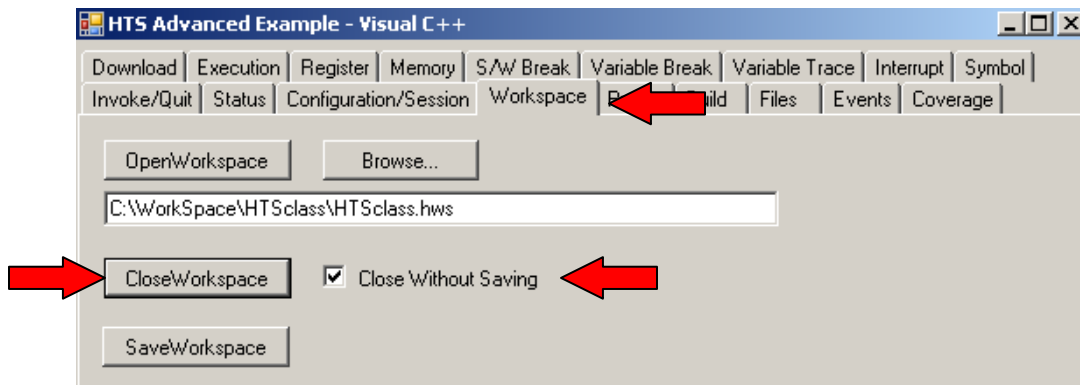
Question: (Write Answers on Sheet at the End of the Lab)

2.3) What does the ResetTargetExec button do?

34. Click the **StopTargetExec** button to stop the target MCU.

35. On the *Workspace* tab, check the **Close Without Saving** option.

36. Click the **CloseWorkspace** button.



37. In *HEW*, verify the workspace has been closed.

38. In the *HTS Advanced Example*, click on the **Invoke/Quit** tab.

39. Click the **QuitHew** button.

40. Verify *HEW* closes.

41. Close the *HTS Advanced Example*.



It may have been a bit cumbersome to jump between the HTS Advanced Application and HEW, but undoubtedly you now understand how HTS can control and manipulate HEW. Given these same commands and a few others under full programmatic control you could easily create an app to download for production or Flash serial numbers into devices. The possibilities are endless.


This is the end of Section 2.

3 Windows programming with HTS – beginning

Time to complete task: 40 minutes

Overview:

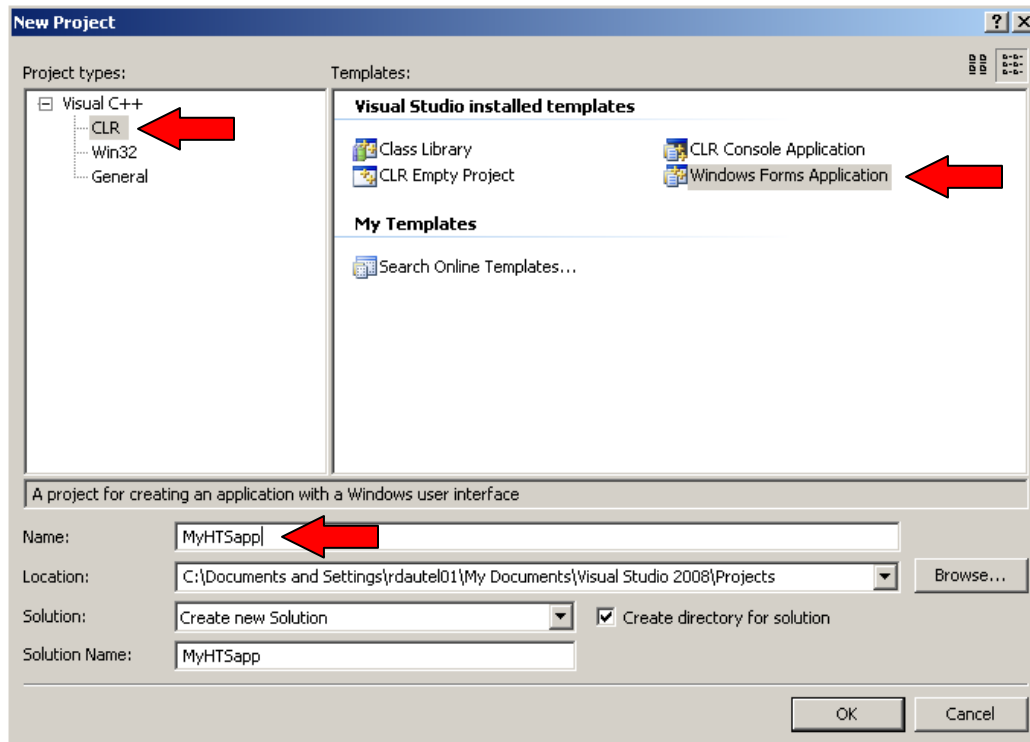
This final lab section walks the user through a procedure to build a simple HTS Windows application in C++ using Visual Studio Express from start to finish.



Note: If you have prior experience using Visual Studio, you may want to work with Section 4 rather than this section.

Procedural Steps:

1. Start Visual C++ Express Edition.
2. Click on **File>>New>>Project**. The *New Project* dialog box will appear.
3. Under the *CLR* Project Types, select **Windows Forms Application**.
4. Name the project **MyHTSapp** and click **OK**.

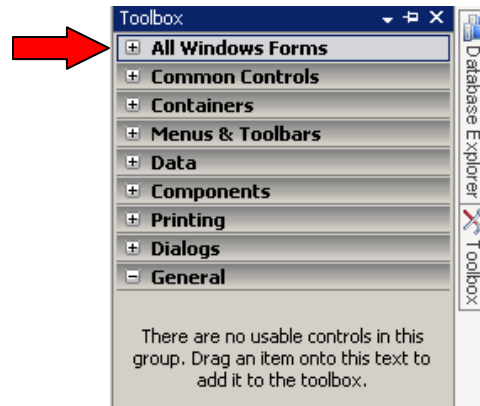


5. Wait a moment for the app to build and the Form Design window will appear.

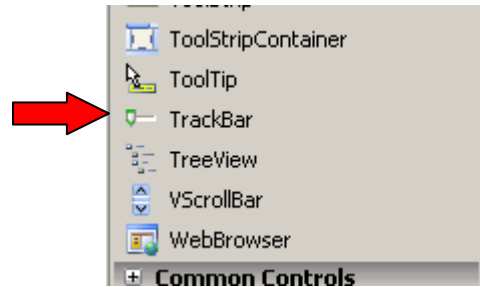
6. On right side is a *Toolbox* tab. Click this to open the *Toolbox*.



7. In the *Toolbox*, expand the **All Windows Forms** section.



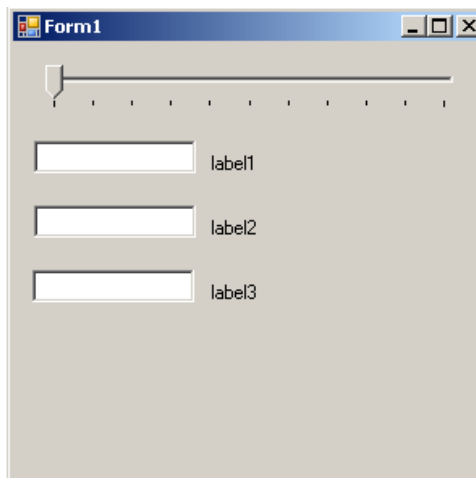
8. Scroll down and find the **TrackBar** component. Drag one of these over to your dialog.



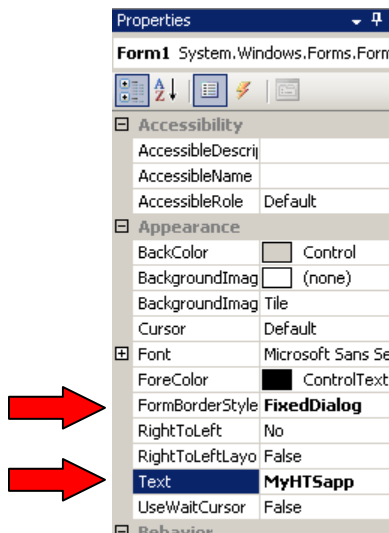
9. Scroll down and find the **TextBox** component. Drag 3 of these over to your dialog. Make sure you keep track of where you place the 1st, 2nd, and 3rd box.

10. In the *ToolBox*, scroll down and find the **Label** component. Drag 3 of these over to your dialog.

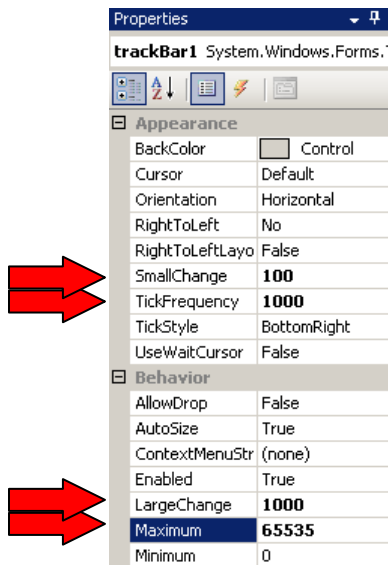
11. Move and size your objects to make your form look something like this:



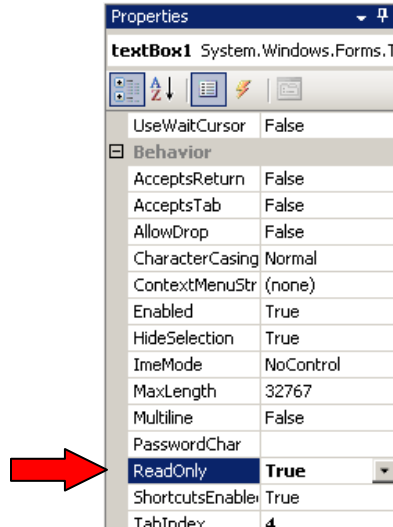
12. On your form, right click and select **Properties**. The *Properties* box will open.
13. Under the *Appearance* section, find the **FormBorderStyle** property and choose **FixedDialog**
14. Under the *Appearance* section, change the Text property from “Form1” to “MyHTSapp”.



15. On your form, click on the *TrackBar* object to change the **Properties Dialog** to the *TrackBar*'s properties.
16. Under the *Appearance* section, find the **SmallChange** property and enter **100**.
17. Under the *Appearance* section, find the **TickFrequency** property and enter **1000**.
18. Under the *Behavior* section, find the **LargeChange** property and enter **1000**.
19. Under the *Behavior* section, find the **Maximum** property and enter **65535**.



20. On your form, click on the first TextBox object to change the *Properties Dialog* to the **TextBox1** properties.
21. In the *Properties* box, under the *Behavior* section, find the **ReadOnly** property and choose **True**.



22. Do the previous two steps for the second and third TextBox objects.
23. On your form, click on the top label, type “**Timer value**”, and press enter.
24. Click on another label and change it to “**MyVar**”.
25. Click on the final label and change it to “**MyVar address**”.
26. Close the *Properties* dialog box.

Note: Don't change the name “MyVar”. The HEW code is using this name.

27. Double click on the *TrackBar* object to open the handler code for the *TrackBar*. You will see the following line of code in a file called **Form1.h**:

```
#pragma endregion
private: System::Void trackBar1_Scroll(System::Object^ sender, System::EventArgs^ e) {
};
```

28. Locate the opening and closing braces for *TrackBar*'s handler function.
29. Enter the following code for the *TrackBar* handler in between the braces:

To keep you from typing a lot of code, we've placed a text file containing the code on the laptop you are using in the *My Documents* folder as file **C05-LabCode.txt**. Just open the text file and the code you need.

```

// We're working with a COM object so it's a good idea to use try/catch.
try
{
    // Setup a data pointer.
    cli::array<unsigned char> ^ data = gcnew cli::array<unsigned char>(2);

    // Get the trackBar's value.
    data[0] = trackBar1->Value & 0xFF;
    data[1] = (trackBar1->Value >> 8) & 0xFF;

    cli::array<unsigned char> ^% tdata = data;

    unsigned int taddr = 0;

    if(nullptr != hts)
    {
        // Use HTS to get the address of the MyVar variable in our HEW project.
        hts->SymbolToAddress("MyVar", taddr);

        // Place the address we got into a textBox
        textBox3->Text = "0x" + String::Format("{0:X}", taddr);

        // Use HTS to set the MCU's TA2 timer register with the TrackBar value.
        hts->SetMemory(0x038A, 0x038B, 1, tdata);

        // Use HTS to set our MyVar variable with the TrackBar value.
        hts->SetMemory(taddr, taddr+1, 1, tdata);

        // Use HTS to read the value of the MCU's TA2 timer register.
        hts->GetMemory(0x038A, 0x038B, 1, tdata);

        // Place the value we got from TA2 into a textBox.
        textBox1->Text = "0x" +
            String::Format("{0:X}", tdata[1]) + String::Format("{0:X}", tdata[0]);

        // Use HTS to read the value of our MyVar variable.
        hts->GetMemory(taddr, taddr+1, 1, tdata);

        // Place the value we got for MyVar into a textBox.
        textBox2->Text = "0x" +
            String::Format("{0:X}", tdata[1]) + String::Format("{0:X}", tdata[0]);
    }
}
// Catch the error is something did not work.
catch (Exception ^ exception)
{
    Console::WriteLine(exception->Message);
}

```

30. Go to the top of the *Form1.h* file and look for this code around line 11:

```
using namespace System::Drawing;
```

31. Under this line, enter:

```
using namespace HEWTARGETSERVERLib;
```

32. Scroll down until you find the top of the class around line 23:

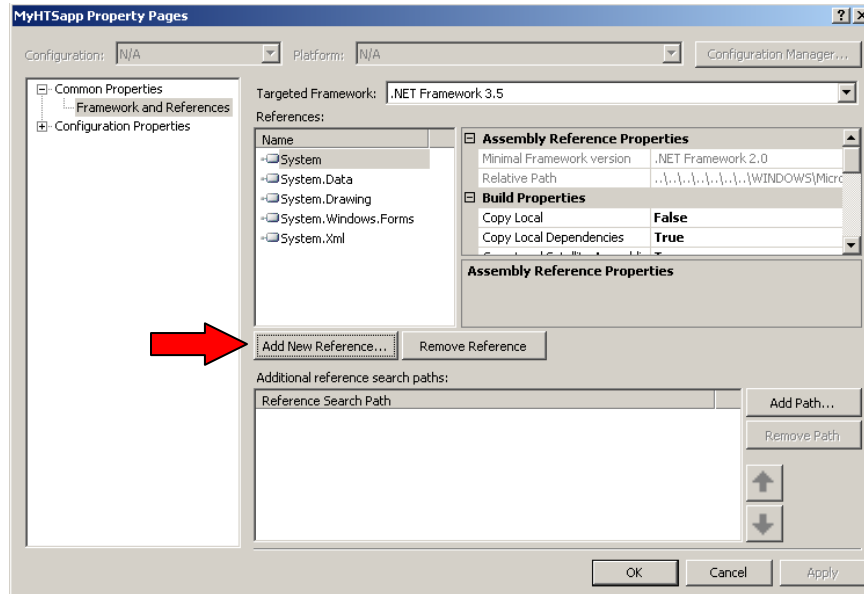
```
public ref class Form1 : public System::Windows::Forms::Form
{
    public:
        Form1(void)
        {
            InitializeComponent();
            //
            //TODO: Add the constructor code here
            //
        }
}
```

33. Modify the code to look like this by adding the lines in bold with the red bar:

```
public ref class Form1 : public System::Windows::Forms::Form
{
    public:
        // Give us a pointer to the HTS object.
        HEWTARGETSERVERLib:HewServer1Class^ hts;
        Form1(void)
        {
            InitializeComponent();
            //
            //TODO: Add the constructor code here
            //
            try //connect to HEW server
            {
                hts = gcnew HEWTARGETSERVERLib:HewServer1Class;
            }
            catch (Exception ^ e)
            {
                String ^ msg = e->Message; // avoid warning
                hts = nullptr;
            }
            if(hts == nullptr)
            {
                String ^ msg = "Missing Target Server Library" +
                    Environment::NewLine;
                MessageBox::Show(msg, "Error", MessageBoxButtons::OK,
                    MessageBoxIcon::Error);
                return;
            }
        }
}
```

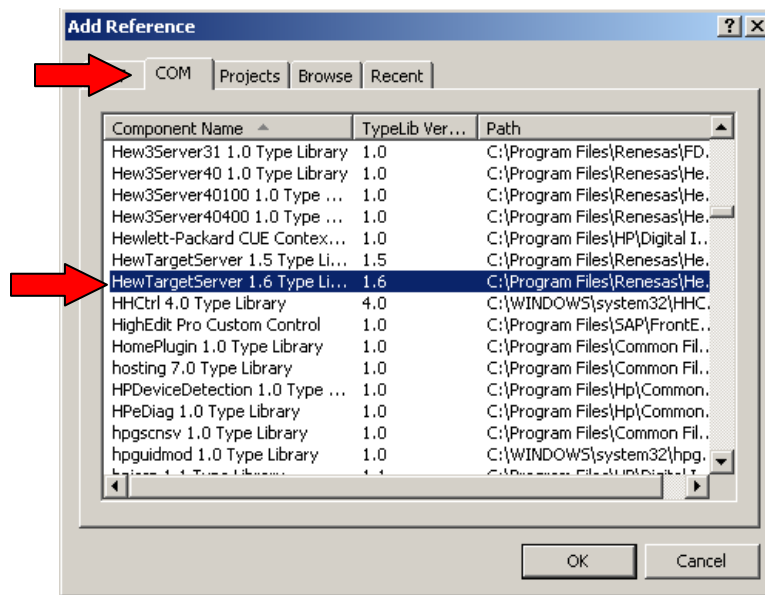
34. On the Visual Studio menu bar, Go to **Project>>MyHTSapp Properties...**

35. On the *MyHTSapp Property Pages*, under the *Common Properties* section, click on the **Add New Reference...** button.



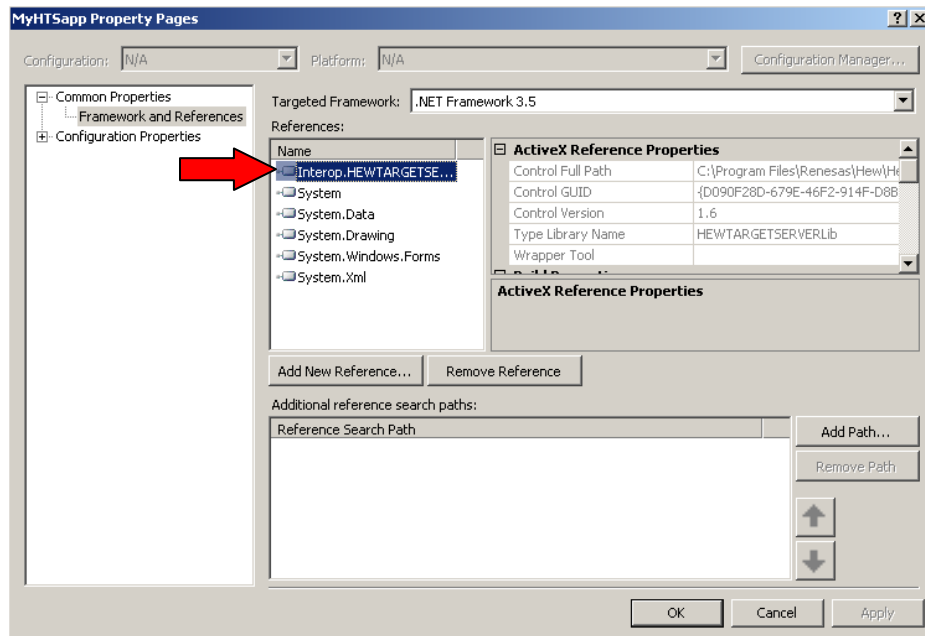
36. In the *Add Reference* dialog box that has opened, click the **COM** tab.


37. Scroll down until you find the component **HewTargetServer 1.6 Type Library**. Select this component and click **OK**.

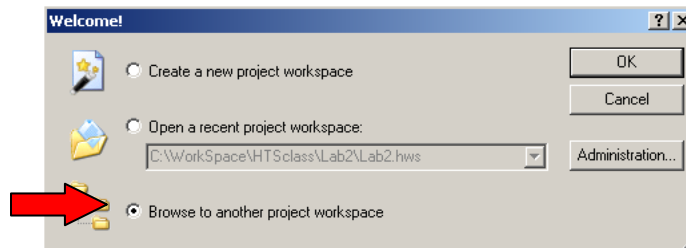


Note: Your list of COM object may look very different from the example pictured above depending on the software installed on your machine.

38. Verify that **Interop.HEWTARGETSERVERLib.1.6** has been added to the *References* list.



39. Click **OK** to close the *MyHTSapp Property Pages* dialog box.
40. Build the program by pressing **F7** or from **Build>>Build Solution**.
41. Click on **Debug>>Start Without Debugging** or press **CTRL+F5** to run your new *MyHTSapp* program.
42. Connect the **HTS board** to the computer.
43. Connect the **headphones** that came in the HTS kit to the *HTS board*.
44. Start **HEW** by double clicking on the HEW icon on the desktop. 
45. In the Welcome dialog box, select **Browse to another project workspace** and click **OK**.




46. In the *Open* dialog box that appears, browse to **C:\WorkSpace\HTSclass\Lab3_4** and select the file:

Lab3_4.hws






Upon opening Lab3_4.hws HEW will load the project, connect to the HTS board, and download the program. Note, this may take several seconds to complete.

47. Run the program on the HTS board by **Debug>>Go** or the  icon in HEW.
48. Jump to the *MyHTSapp* program and move it to the upper left corner of the screen so you can see the MyHTSapp program with HEW underneath it.
49. With HEW running, move the **TrackBar** in the *MyHTSapp* program. Verify the sound on the headphones changes along with an LED on the board.
50. In the *MyHTSapp* program, verify **Timer value** and **MyVar** are updating as the TrackBar is moved. Note, you will not see values until the TrackBar is moved.



The function that handles the slider (TrackBar component) uses several HTS commands.

- SetMemory is used to poke data into the target MCU's memory map. This can be ANYTHING that is mapped. Registers, internal memory, or external memory.
- GetMemory is the opposite of SetMemory and is used to peek data in the target MCU's memory map. It too, can work on anything from registers to external memory.
- SymbolToAddr is an extremely useful function that converts debug symbols to address values provided the symbols are loaded into HEW.

51. In *Visual Studio*, scroll down in *Form1.h* to about line 194 to view the TrackBar's handler code.
52. Spend a few minutes reading over the code to see what it does while you experiment with the MyHTSapp's controls and HEW. You may also want to start, stop, or reset the MCU in HEW using the following commands:
 - Run the MCU by **Debug>>Go** or the  icon.
 - Stop the MCU by **Debug>>Halt Program** or the  icon.
 - Reset the MCU by **Debug>>Reset CPU, F5**, or the  icon.

Question: (Write Answers on Sheet at the End of the Lab)

3.1) In the MyHTSapp program, why are the *Timer value* and *MyVar* values different?

Question: (Write Answers on Sheet at the End of the Lab)



3.2) Why is the *SymbolToAddress* function called first?

Question: (Write Answers on Sheet at the End of the Lab)

3.3) Why does one SetMemory/GetMemory use a hardcoded address of 0x038A-0x038B while the other pair uses the variable taddr?

Question: (Write Answers on Sheet at the End of the Lab)

3.4) Why does the board continue to make sound when the MCU's program has been stopped?

53. In *HEW*, hit the **Stop**  button and disconnect *HEW* from the *HTS board* by clicking on the **Disconnect**  icon or by **Debug>>Disconnect**.
54. Close *HEW*, Visual Studio, and *MyHTSapp*.

This is the end of Section 3.

4

Windows programming with HTS – advanced

Time to complete task: 40 minutes

Overview:

This final lab section points out some of the actual Windows functions used in a simple HTS Windows application built in C++ using Visual Studio Express. We will begin with installing the HTS COM object into the project, then finish by looking at some of the HTS functions and events.



Note: If you do not have prior experience using Visual Studio or building Windows applications, you may want to work with Section 3 rather than this section.

Procedural Steps:

1. Start Visual C++ Express Edition.
2. Click on **File>>Open>>Project/Solution..**
3. Select **HTSapp** and open the project.
4. In the project tree, right click on *Form1.h* and select **View Code**.
5. Look for the following code around line 16:


```
using namespace System::Drawing;
```
6. Under this line, add:

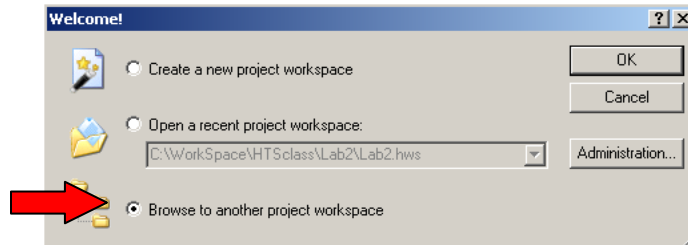

```
using namespace HEWTARGETSERVERLib;
```
7. Go to **Project>>HTSapp Properties...**
8. In the *HTSapp Property Pages*, under the *Common Properties* section, click on the **Add New Reference...** button.
9. In the *Add Reference* dialog box that has opened, click the **COM** tab.
10. Scroll down until you find the component **HewTargetServer 1.6 Type Library**. Select this component and click **OK**.
11. Verify that **Interop.HEWTARGETSERVERLib.1.6** has been added to the *References* list.
12. Click **OK** to close the *HTSapp Property Pages* dialog box.
13. Build the program by pressing **F7** or from **Build>>Build Solution**.
14. Click on **Debug>>Start Without Debugging** or press **CTRL+F5** to run the *HTSapp*.
15. Connect the **HTS board** to the computer.

16. Connect the **headphones** that came in the HTS kit to the *HTS board*.

17. Start **HEW** by double clicking on the HEW icon on the desktop.



18. In the Welcome dialog box, select **Browse to another project workspace** and click **OK**.




19. In the *Open* dialog box that appears, browse to **C:\Workspace\HTSclass\Lab3_4** and select the file:

Lab3_4.hws



Upon opening Lab3_4.hws HEW will load the project, connect to the HTS board, and download the program. Note, this may take several seconds to complete.

20. Run the program on the HTS board by **Debug>>Go** or the  icon in HEW.

21. Jump to the *HTSapp* and move it to the upper left corner of the screen so you can see the HTSapp with HEW underneath it.

22. Click on the **Stop/Go** button. Verify nothing happens.

23. In *HEW*, stop the MCU by **Debug>>Halt Program** or the  icon.

24. Close the *HTSapp* program.



In order for the Stop/Go button to work we need to add some HTS commands to the Windows function that handles the button presses. The HTS function StopTargetExec() tells HEW to halt the target MCU while HTS function GoTargetExec() tells HEW to start the target MCU at its current PC position. Both commands are preceded by a simple test to ensure we have a valid pointer to the HTS COM object.

25. In the project tree in *Visual Studio*, double click **Form1.cpp** to open that file.

26. Scroll down to about line 332.

27. Uncomment the following line to add the StopTargetExec() HTS function to the button's code.

```
// UNCOMMENT    if(nullptr != hts) hts->StopTargetExec();
```

28. Scroll down a few more lines and uncomment the following line to add the GoTargetExec() HTS function to the button's code.

```
// UNCOMMENT    if(nullptr != hts) hts->GoTargetExec();
```



If you examine the button's code you will notice two other features.

- First, we have a flag called `IsRunning` which determines the HTS function the button's press will execute. We'll talk more about `IsRunning` in a few steps.
- Second is that both HTS functions are encapsulated in a try/catch command in order to process any HTS errors which may occur.

29. Rebuild the program by pressing **F7** or from **Build>>Build Solution**.

30. Click on **Debug>>Start Without Debugging** or press **CTRL+F5** to run the *HTSapp*.

31. In *HEW*, run the program on the HTS board by **Debug>>Go** or the  icon in *HEW*.

32. In the *HTSapp* program, click on the **Stop/Go** button. Look at *HEW* to verify it is responding to these commands.

33. With the target application running, move the **TrackBar** in the *HTSapp* program. Verify the sound on the headphones changes along with and LED on the HTS board.

34. In the *HTSapp* program, verify **Timer value** and **MyVar** are updating as the **TrackBar** is moved.






The function that handles the **TrackBar** component uses several HTS commands:

- `SetMemory` is used to write data into the target MCU's memory map. This can be ANYTHING that is mapped. Registers, internal memory, or external memory.
- `GetMemory` is the opposite of `SetMemory` and is used to read data in the target MCU's memory map. It too, can work on anything from registers to external memory.
- `SymbolToAddr` is an extremely useful function that converts debug symbols to address values provided the symbols are loaded into *HEW*.

35. In *Visual Studio*, scroll down in *Form1.cpp* to about line 295.

36. Spend a few minutes reading over the code to see what it does while you experiment with the *HTSapp*'s controls and *HEW*. You may also want to start, stop, or reset the MCU in *HEW* using the following commands:

- Run the MCU by **Debug>>Go** or the  icon.
- Stop the MCU by **Debug>>Halt Program** or the  icon.
- Reset the MCU by **Debug>>Reset CPU, F5**, or the  icon.

Question: (Write Answers on Sheet at the End of the Lab)

4.1) In the HTSapp2 program, why are the *Timer value* and *MyVar* values different?

Question: (Write Answers on Sheet at the End of the Lab)

4.2) Why is the *SymbolToAddress* function called first?

Question: (Write Answers on Sheet at the End of the Lab)

4.3) Why does one *SetMemory/GetMemory* use a hardcoded address of 0x038A-0x038B while the other pair uses the variable *taddr*?

Question: (Write Answers on Sheet at the End of the Lab)

4.4) Why does the board continue to make sound when the MCU's program has been stopped?

37. In *Visual Studio*, scroll up to about line 38 in *Form1.cpp* and read over the setup code for the two HTS events, *Go* and *Stop*.





While very different from the way a message handler looks in MFC or Win32 programming, these two functions point the HTS's event handles for the *Go* and *Stop* events to functions written to handle them in our code. HTS currently has 13 different event messages. More info on the different HTS events can be found in the *HewTargetServer User's Manual*.

38. In *Visual Studio*, scroll down to about line 65 in *Form1.cpp* and look over the actual event handler code for the *Go* and *Stop* events.

Question: (Write Answers on Sheet at the End of the Lab)

4.5) What causes the boxes on the HTSapp program to change color?

39. In *HEW*, hit the **Stop**  button and disconnect *HEW* from the *HTS board* by clicking on the **Disconnect**  icon or by **Debug>>Disconnect**.

40. In the *HTSapp* program, press the **Start/Stop** button or move the **TrackBar**. Notice the error message.

Question: (Write Answers on Sheet at the End of the Lab)

4.6) Why was an error generated when *HEW* was disconnected from the *HTS board*?



HTS commands such as GetMemory and SetMemory require a connection to the target MCU whereas HTS commands such as OpenWorkspace and GetStatus only require HEW to be running. Luckily, there are two events, Link Up and Link Down which report when a target board has been connected or disconnect. Due to the simplicity of this project, those events were not setup and we simply rely on the error message. An actual program might take advantage of HTS and autoconnect to a board and use the Link events to verify a successful operation. You can also poll status by using one of several HTS status functions.

41. Close HEW, Visual Studio, and HTSapp.

This is the end of Section 4.

Answer Page

Section 1: Setting up HTS.

No questions.

Section 2: Getting familiar with HTS.

2.1) Why did the *Confirmation Request* message appear?

2.2) What is the purpose of the *IsRunning* button?

2.3) What does the *ResetTargetExec* button do?

Answer Page

Section 3: Windows programming with HTS – beginning

3.1) In the MyHTSapp program, why are the *Timer value* and *MyVar* values different?

3.2) Why is the *SymbolToAddress* function called first?

3.3) Why does one SetMemory/GetMemory use a hardcoded address of 0x038A-0x038B while the other pair uses the variable taddr?

3.4) Why does the board continue to make sound when the MCU's program has been stopped?

Answer Page

Section 4: Windows programming with HTS – advanced

4.1) In the HTSapp2 program, why are the *Timer value* and *MyVar* values different?

4.2) Why is the *SymbolToAddress* function called first?

4.3) Why does one *SetMemory/GetMemory* use a hardcoded address of 0x038A-0x038B while the other pair uses the variable *taddr*?

4.4) Why does the board continue to make sound when the MCU's program has been stopped?

4.5) What causes the boxes on the HTSapp program to change color?

4.6) Why was an error generated when HEW was disconnected from the HTS board?
