

Gathering and Modifying Real-Time Software Data in a Visual Studio Environment

Mark Rogers

November 6, 2008

Executive Summary

The topics discussed in this application note include gathering real-time software data, performing calculations on it, and writing it back to memory. This will be explained in the context of Microsoft's Visual Studio .Net software suite. Code will be presented to accurately explain how this will be implemented.

Keywords

C#, C++, DLL, reference, function, real-time, interface

Introduction

Real-time application software is mostly used in simulation and gaming applications. The market for computer simulation and gaming continues to grow. As the market grows, the need to be able to interact with software data will continue to grow as well. In this application note I will discuss how to access simulation data, perform any necessary calculations or changes to it, and then write it back to memory in real time. Speed is always an issue with real-time software. This will be taken into account during the course of the note. Both C# and C++ programming languages will be used throughout this application note. C++ is used to create the DLL and C# is used to create the actual data interface. This will be explained in more detail later on.

Objective

The purpose of this application note is to describe the process of accessing, modifying, and writing back real-time software data. The process will be described in the following series of steps:

1. Creation of DLL Project in C++
2. Writing Code for the DLL in C++
3. Creation of Interface Project in C#
4. Declaration of DLL in C#
5. Assessing Data

6. Modifying Data
7. Writing Data Back to Proper Memory Address

Steps

1. Creation of DLL Project in C++

A DLL is a Dynamically Linked Library. It is a library that is linked to application programs when they are loaded or running. In other words, it can be used as a way of accessing program data. Hopefully the software application that the programmer is working with has some mechanism of getting and setting a shared data structure. In this tutorial I will use an example that I have made up that accurately explains how a DLL can be created.

To create the DLL project click File -> New -> Project and select MFC DLL as shown below. Then, select a name for the DLL you wish to create.

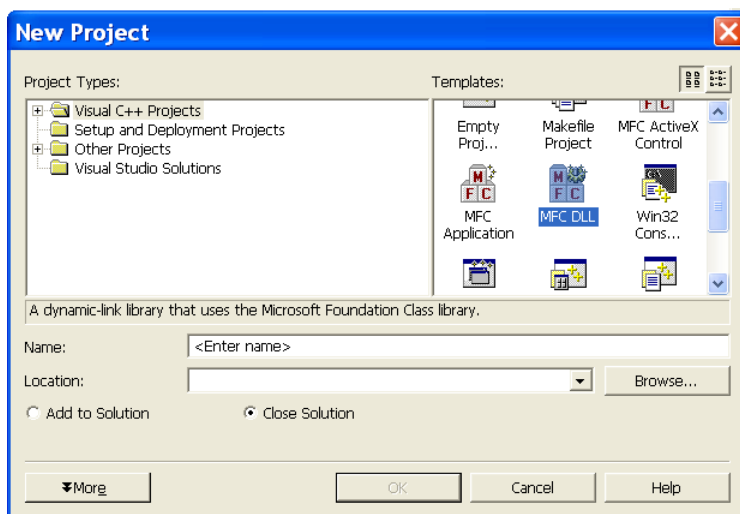


Figure 1: Creation of DLL project in C++

2. Writing Code for the DLL in C++

Hopefully, the developer of the software application you are working with has defined a function in which the shared data structure can be set. Inside of the .cpp file, the library windows.h should be included at the top of the code file. Also, your header file for the shared data structure should be included at the top of the code file. Below is example code for what I explained.

```

#include <windows.h>
#include <stdio.h>
#include <stdlib.h>

#include "SharedStructure.h"

struct Shared_Data {

    float speed;
    float distance;
    float height;
    float direction;
    float score;|

    long time;
    bool isCurrentlyActive;

} *SharedData;

SharedStructure set_SharedData( (void**) (&SharedData));

```

← This is the function provided by the software.

Figure 2: Include statements and structure code for DLL

There are four include statements. The final one is the include statement for the shared structure header file. This file would most likely be provided with the software application that one is working with. I have defined a sample Shared_Data structure as a pointer. The variables inside the structure are used by the software application while it is running. The final line is the set_SharedData function. This would also be provided by the developer of the software application. This lets the software know what shared data structure to use. The “&” before SharedData indicates that the shared data structure is passed by reference. Passing a parameter by reference means that the parameter can change during the course of the function call.

```

VOID WINAPI get_shared_data( float& speed, float& distance, float& height, float& direction, float& score)
{
    if (SharedData->isCurrentlyActive)
    {
        speed = SharedData->speed;
        distance = SharedData->distance;
        height = SharedData->height;
        direction = SharedData->direction;
        score = SharedData->score;
    }
}

VOID WINAPI set_shared_data( float& speed, float& distance, float& height, float& direction, float& score)
{
    if (SharedData->isCurrentlyActive)
    {
        SharedData->speed = speed;
        SharedData->distance = distance;
        SharedData->height = height;
        SharedData->direction = direction;
        SharedData->score = score;
    }
}

```

Figure 3: DLL get and set functions

The above picture shows the two DLL functions that are defined. The `get_shared_data` function retrieves the shared data register values. The `set_shared_data` function sets the shared data register values. The values that are retrieved and that the shared registers are set to are passed by reference to the function. In C++, the “->” operator is used to access a field of an object that is a pointer. The `isCurrentlyActive` field of the `SharedData` structure is set to true when the application software is active and false when the application software is not active. Also, the time field within the `SharedData` structure is also set by the application software.

3. Creation of Interface Project in C#

In Microsoft Visual C#, go to File -> New Project. Next, select a Windows Forms Application, and then type in your interface project name as shown below.

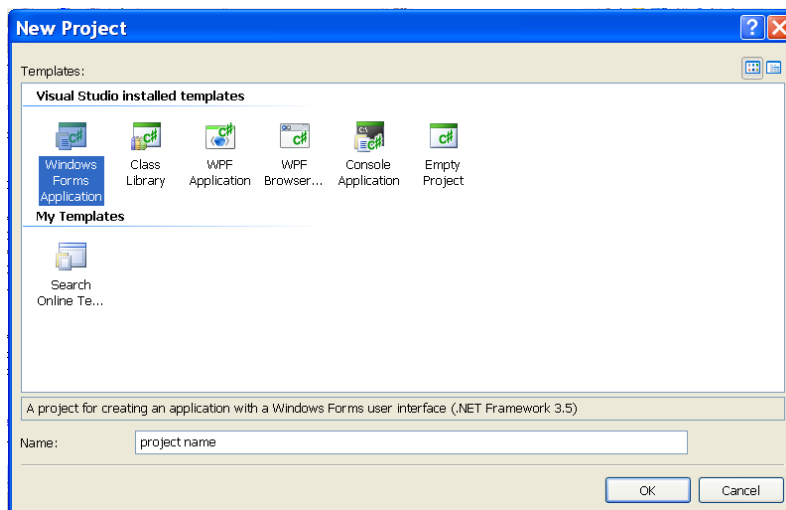


Figure 4: Creation of interface project in C#

4. Declaration of DLL in C#

In this step, the C++ DLL that was created in step 2 will be declared in our C# interface. The picture below shows how this can be done. The code seen below is inside of `Form1.cpp`. The using directive `System.Runtime.InteropServices` should be in the header in order to be able to reference a C++ DLL. The two DLL functions that are used should be declared as shown below. The red print in the code below indicates where the C++ DLL is located in the current machine's file system. The “\\” indicates folder hierarchy in C#. After the `DllImport` line, the next line declares the particular DLL function that will be used during the course of the program. Notice that the function name matches exactly what the DLL function was named. Also notice that all

parameters in the function have the ref keyword before them. This means that they are passed by reference.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Runtime.InteropServices;

namespace sample
{
    public partial class Form1 : Form
    {
        [DllImport("C:\\sample\\shared.dll", CharSet = CharSet.Unicode)]
        public static extern void get_shared_data(ref float speed, ref float distance, ref float height,
            ref float direction, ref float score);
        [DllImport("C:\\sample\\shared.dll", CharSet = CharSet.Unicode)]
        public static extern void set_shared_data(ref float speed, ref float distance, ref float height,
            ref float direction, ref float score);

        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

Include this using directive

Use this format for DLL function declaration

Figure 5: Referencing C++ DLL in C#

5. Assessing Data

Since interacting with a real-time software application is taking place, data will need to be accessed at specific time intervals. In order to do this, a timer control must be added to the interface application. This can be done by going to View->Toolbox and then scrolling down to find a control called “Timer.” Drag and drop this onto Form1.cs [Design].

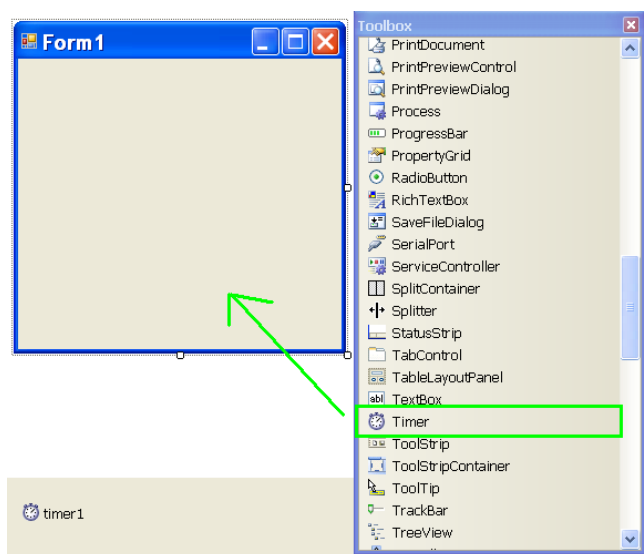


Figure 6: Timer control

In order to use the DLL function and access the data, double click the timer1 control and put the following code inside the `timer1_Tick(object sender, EventArgs e)` code as shown below. The `get_shared_data` function will update the five data variables passed into it.

```
private void timer1_Tick(object sender, EventArgs e)
{
    float speed = 0.0f;
    float distance = 0.0f;
    float height = 0.0f;
    float direction = 0.0f;
    float score = 0.0f;

    get_shared_data(ref speed, ref distance, ref height, ref direction, ref score);
}
```

Figure 7: Using `get_shared_data` function

6. Modifying Data

In the code shown below, the data that was grabbed from the real-time application, through the DLL is modified. Distance and height are updated based on speed. The score is also updated if speed is greater than 0.

```
private void timer1_Tick(object sender, EventArgs e)
{
    float speed = 0.0f;
    float distance = 0.0f;
    float height = 0.0f;
    float direction = 0.0f;
    float score = 0.0f;

    get_shared_data(ref speed, ref distance, ref height, ref direction, ref score);

    distance += speed;
    height += 0.5f * speed;
    if (speed > 0)
    {
        score += 5.0f;
    }
}
```

Figure 8: Modifying data

7. Writing Data Back to Proper Memory Address

In order for the changes to the data to take affect in the real-time software application, the data must be written back to the proper memory place. This is shown below in the code. The `set_shared_data` function is called with the five data parameters.

```

private void timer1_Tick(object sender, EventArgs e)
{
    float speed = 0.0f;
    float distance = 0.0f;
    float height = 0.0f;
    float direction = 0.0f;
    float score = 0.0f;

    get_shared_data(ref speed, ref distance, ref height, ref direction, ref score);

    distance += speed;
    height += 0.5f * speed;
    if (speed > 0)
    {
        score += 5.0f;
    }

    set_shared_data(ref speed, ref distance, ref height, ref direction, ref score);
}

```

Figure 9: Using set_shared_data function

Also, in order to control on what interval the data will be updated, the timer control has an interval property as shown below. A higher interval indicates less updates, while a lower interval indicates more updates. The interval can be controlled to what best suits the programmer's needs.

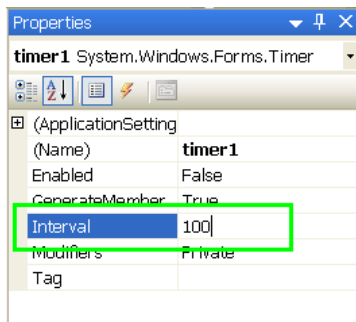


Figure 10: Setting timer interval

Results

The result is that the real-time software application data is updated every hundred timer ticks. This interval can be changed as shown above.

Conclusion

The steps taken to create a DLL and corresponding interface are simple, especially if one has knowledge of the C++ and C# programming languages. Variables are passed by reference so that they may be changed during the course of a function. Real-time software application data can be interacted with fairly easily especially if the software application has built in shared data structures.

References

“DllImportAttribute Class.” MSDN: .Net Framework Developer Center. 06 Nov. 2008

<[http://msdn.microsoft.com/en-](http://msdn.microsoft.com/en-us/library/system.runtime.interopservices.dllimportattribute.aspx)

[us/library/system.runtime.interopservices.dllimportattribute.aspx](http://msdn.microsoft.com/en-us/library/system.runtime.interopservices.dllimportattribute.aspx)>.

“Visual C++ DLLs.” MSDN: Visual C++ Developer Center. 06 Nov. 2008

<[http://msdn.microsoft.com/en-us/library/1ez7dh12\(VS.80\).aspx](http://msdn.microsoft.com/en-us/library/1ez7dh12(VS.80).aspx)>.