

**TREK**

**HOW TO BUILD A VISUAL C++ DISPLAY**

**TUTORIAL**



**November 2012**

Approved for Public Release; Distribution is Unlimited.

## TABLE OF CONTENTS

<u>PARAGRAPH</u>	<u>PAGE</u>
<b>1 What You Need To Know Before You Read This Document.....</b>	<b>1</b>
<b>2 Technical Support.....</b>	<b>1</b>
<b>3 Introduction.....</b>	<b>2</b>
<b>4 Step-By-Step .....</b>	<b>5</b>
<b>5 Some Final Notes About This Tutorial .....</b>	<b>58</b>
<b>Appendix A Glossary.....</b>	<b>60</b>
<b>Appendix B Acronyms.....</b>	<b>67</b>

## FIGURES

<u>FIGURE</u>	<u>PAGE</u>
Figure 1 Cyclic Application Main Window .....	2
Figure 2 Conceptual Design of the Cyclic Application .....	3
Figure 3 Cyclic Application Threads.....	3
Figure 4 Scenario.....	4
Figure 5 New Dialog Box.....	8
Figure 6 MFC AppWizard - Step 1 .....	9
Figure 7 MFC AppWizard – Step 2.....	10
Figure 8 MFC AppWizard – Step 3.....	11
Figure 9 MFC AppWizard – Step 4.....	12
Figure 10 MFC AppWizard – Step 5.....	13
Figure 11 MFC AppWizard – Step 6.....	14
Figure 12 MFC AppWizard - Step 6 Dialog showing Base class CFormView Selection. ....	15
Figure 13 New Project Information dialog.....	16
Figure 14 Cyclic Application Main Window .....	18
Figure 15 Visual C++ Resource Tab .....	20
Figure 16 The IDD_CYCLIC_FORM after the controls have been placed on the form. ....	21
Figure 17 Static Control properties dialog.....	22
Figure 18 Static control properties dialog showing MSID038 caption. ....	22
Figure 19 Cyclic Main Window Controls .....	23
Figure 20 Edit Control Before Control ID Change.....	24
Figure 21 Edit Control After Control ID Change. ....	24
Figure 22 ClassWizard Dialog Before Adding Member Variables .....	25
Figure 23 Add Member Variable for IDC_MSID038_INT_API_EDIT .....	26
Figure 24 Add Member Variable for IDC_MSID038_INT_STATUS_EDIT .....	26
Figure 25 Add Member Variable for IDC_MSID038_INT_VALUE_EDIT .....	27
Figure 26 Class Wizard after adding Member Variables.....	28
Figure 27 ClassWizard Dialog with the Message Maps tab selected. ....	29
Figure 28 ClassWizard OnUpdate Message .....	30
Figure 29 Cyclic main window with static and edit controls.....	31
Figure 30 Visual C++ Menu Resource Editor .....	32
Figure 31 Edit Menu Deletion Warning. ....	33
Figure 32 Menu Resource Editor showing Empty Menu Frame .....	34
Figure 34 Cyclic Application with Update Menu.....	35
Figure 33 Cyclic Menu Bar with Update Menu .....	35
Figure 35 Update Menu with Start Item .....	36
Figure 36 Update Menu with Stop Item .....	36
Figure 37 Add Member Function message dialog.....	37
Figure 38 Add Member Function dialog. ....	38
Figure 39 Cyclic Application with Update Menu.....	39
Figure 40 Insert Files into Project Dialog.....	41
Figure 41 Cyclic project showing the project files list. ....	42
Figure 42 Project Settings Dialog After Modifications. ....	51
Figure 43 Project Settings Dialog showing reference to TReK API Library. ....	52
Figure 44 Options Dialog. ....	58

## 1 What You Need To Know Before You Read This Document

This tutorial assumes the following:

- You are familiar with the material in the TReK Getting Started User Guide (TREK-USER-001) and the TReK Telemetry Tutorial (TREK-USER-002).
- You are familiar with the following material in the TReK Telemetry Application Programming Interface Reference Manual (TREK-USER-027):
  - \* Sections 1 – 8
  - \* GetOneNewestConvertedIntegerValue Function Description
- You are an average C or C++ programmer.
- You are familiar with Microsoft Visual C++ (version 6.0). You know how to use the AppWizard, the Class Wizard, the Resource Editor, Messages, and Member Variables. You are familiar with the Single Document Interface and know the difference between Document files and View files. You are familiar with the concept of threads (but you don't necessarily know how to use them).
- You know how to start the TReK Telemetry Processing application, add a packet to the packet list, and activate the packet. (See TReK Telemetry Processing User Guide TREK-USER-003.)
- You know how to start the TReK Training Simulator application, add a packet to the packet list, and send the packet. (See TReK Training Simulator User Guide TREK-USER-004.)

## 2 Technical Support

If you are having trouble installing the TReK software or using any of the TReK software applications, please try the following suggestions:

Read the appropriate material in the manual and/or on-line help.

Ensure that you are correctly following all instructions.

Checkout the TReK Web site at <http://trek.msfc.nasa.gov/> for Frequently Asked Questions.

If you are still unable to resolve your difficulty, please contact us for technical assistance:

TReK Help Desk E-Mail, Phone & Fax:

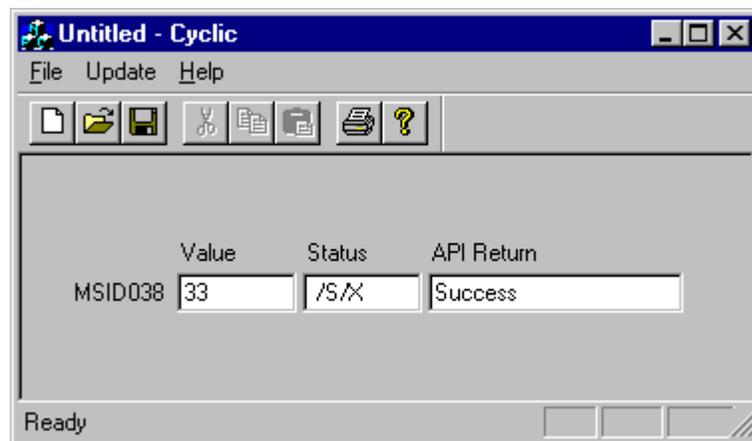
E-Mail:            trek.help@nasa.gov  
 Telephone:        256-544-3521 (8:00 a.m. - 4:30 p.m. Central Time)  
 Fax:                256-544-9353

TReK Help Desk hours are 8:00 a.m. – 4:30 p.m. Central Time Monday through Friday. If you call the TReK Help Desk and you get a recording please leave a message and someone will return your call. E-mail is the preferred contact method for help. The e-mail message is automatically forwarded to the TReK developers and helps cut the response time.

### 3 Introduction

This tutorial will walk you through the process of building a Visual C++ application. This application will use the TReK Application Programming Interface (API) to retrieve telemetry data and display the data in the main window of the application. The application is named Cyclic because the telemetry data will be updated once every second in a “cyclic” fashion.

The Cyclic application is a Single Document Interface application. Figure 1 shows the Cyclic main window.

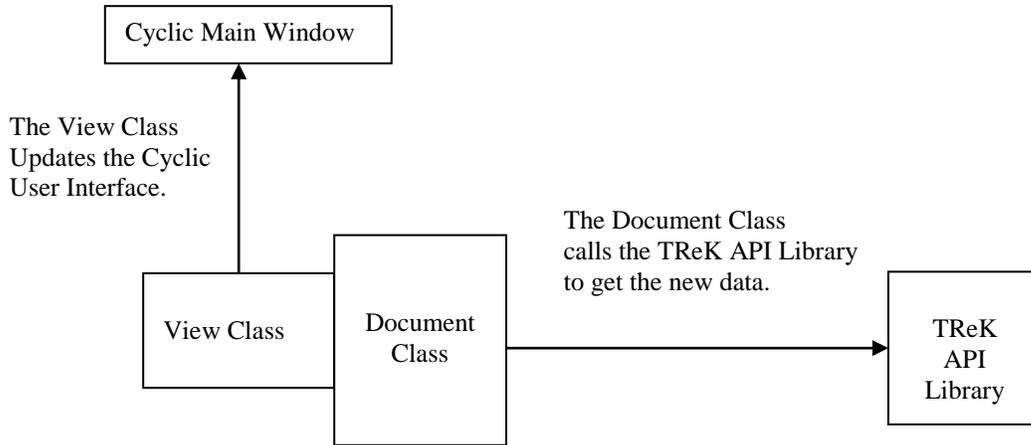


**Figure 1 Cyclic Application Main Window**

The Update menu contains two menu items: Start and Stop. When the user selects Start the values for MSID038 in the main window are updated once every second. To stop the display from updating the user can select Stop.

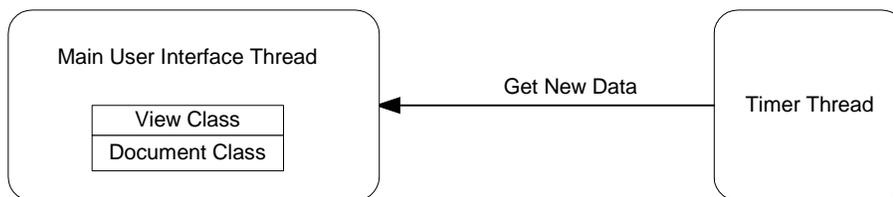
Figure 2 shows a conceptual design of the Cyclic application. The View class is responsible for updating the Cyclic user interface. The Document class is responsible for doing the application’s main work. The work in this case consists of getting new

telemetry data once every second. The Document class interfaces with the TReK API library in order to access the real-time telemetry data.



**Figure 2 Conceptual Design of the Cyclic Application**

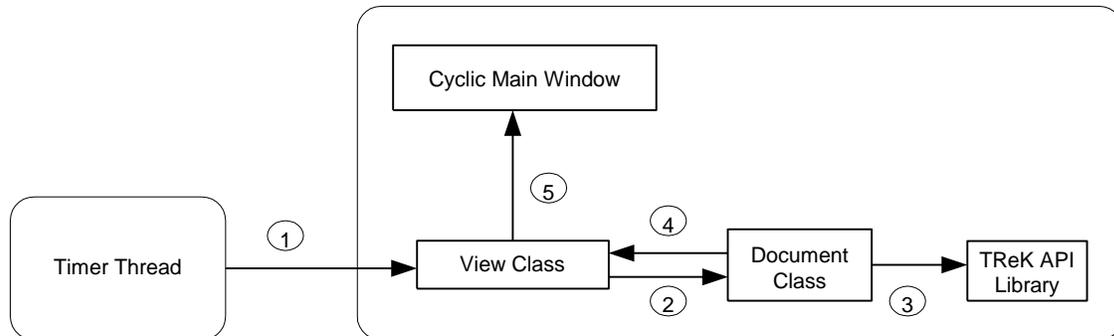
There are two threads in the Cyclic Application. The Main user interface thread and another thread that serves as a timer.



**Figure 3 Cyclic Application Threads**

When the user selects the Start menu item, the second thread (the timer thread) is created. The timer thread is very simple. It simply posts a message to the view class once a second.

Figure 4 shows a step-by-step scenario of what occurs when the view class receives this message.



**Figure 4 Scenario**

The circled numbers in the figure correspond to the following steps:

- Step 1: The Timer Thread sends a message to the View Class.
- Step 2: When the View Class receives this message it calls the Document Class.
- Step 3: The Document Class calls the TReK API to get the new data and then updates the document class member variables that hold that new data.
- Step 4: The document class then tells the View Class to update the main window with the new data.
- Step 5: The View class updates the controls in the main window to display the new data.

This sequence of events occurs over and over again (once a second) until the user selects Stop from the Update menu. When the user selects Stop, the second thread is terminated and the data in the Cyclic main window stops updating. If the user selects Start again, a new timer thread is created and the entire sequence starts over again and continues until the user selects stop.

The next section discusses how to build the Cyclic application step-by-step.

## 4 Step-By-Step

The Step-By-Step instructions have been divided into 4 parts so that there are natural stopping points along the way. Although you don't have to do all four parts in one sitting you need to do them in order. The four parts are as follows:

Part I: Creating the Visual C++ Cyclic Project.

During this part of the tutorial you will use the AppWizard to generate the Single Document Interface files.

Part II: Adding the Menus and Controls to the Main Window.

During this part of the tutorial you will add the menus and controls to the Cyclic main window. You will also define the messages and member variables used in the view class and document class.

Part III: Adding the Timer Thread.

During this part of the tutorial you will add the timer thread. If you aren't familiar with threads don't worry. The code needed to implement the timer thread has been provided for you and can be copied out of the TReK installation directory. This part of the tutorial will explain how to add the timer thread to your project and how to use it. Here's some information about the files that implement the timer thread. There are four files that you will copy out of the TReK installation directory. Each one is described below.

UpdateThread.h	The timer thread is implemented using a class that is derived from CWinThread. The UpdateThread.h file contains the class definition for the timer thread class (which is called CUpdateThread).
UpdateThread.cpp	The UpdateThread.cpp file contains the implementation of the timer thread. The timer thread is very simple. It simply posts a message to the View class once every second. When you create the thread you will pass in an update rate that defines how often the thread will post the message to the View class. For this tutorial you will set the update rate to 1000 milliseconds.
UpdateInfo.h	When you set up the thread, there are several pieces of information that you need to pass in to the thread so it can do its job (such as a handle to the View class and the update rate). The UpdateInfo.h file contains a class definition for an object (called CUpdateInfo) that is used to hold this information.
UpdateInfo.cpp	The UpdateInfo.cpp file contains an empty constructor and destructor for the UpdateInfo.h file.

#### Part IV: Adding the Call to the TReK Application Programming Interface Library.

During this part of the tutorial you add a reference to the TReK Application Programming Interface library so it will be linked into your Cyclic project. This part of the tutorial also shows you how to add the TReK API call to the document class.

The Cyclic project files that match the finished version of this tutorial can be found in the TReK installation directory in the examples\Visual C++\Cyclic directory. These files can be a good resource if you want to copy and paste code as you work along instead of typing it in from scratch. You can also open this document in Microsoft Word and copy the code sections straight from Microsoft Word into the Visual C++ editor. The files in the examples directory also provide an easy way to verify that you have entered the correct information. For example if you run into a compile error, check the example files and compare them to your own.

Remember to perform incremental saves as you work through the tutorial. You never know when there's going to be a power outage. ☺

Well that's about it – Have Fun!

## **Part I**

### **Creating the Visual C++ Cyclic Project**

1. Start the Visual C++ Application.
2. Go to the **File** menu and select **New...**
3. In the New dialog select the **Projects** tab. On the left side of the Projects tab there is a list of project types. You will be creating an MFC application. Select **MFC AppWizard (exe)**. On the right side of the dialog you must enter the **Project name** (Cyclic) and **Location** (D:\Cyclic). (Note: You can of course enter any location you'd like – this is just an example). After you have entered this information your dialog should look like the one in Figure 5. Once you are finished push **OK**.

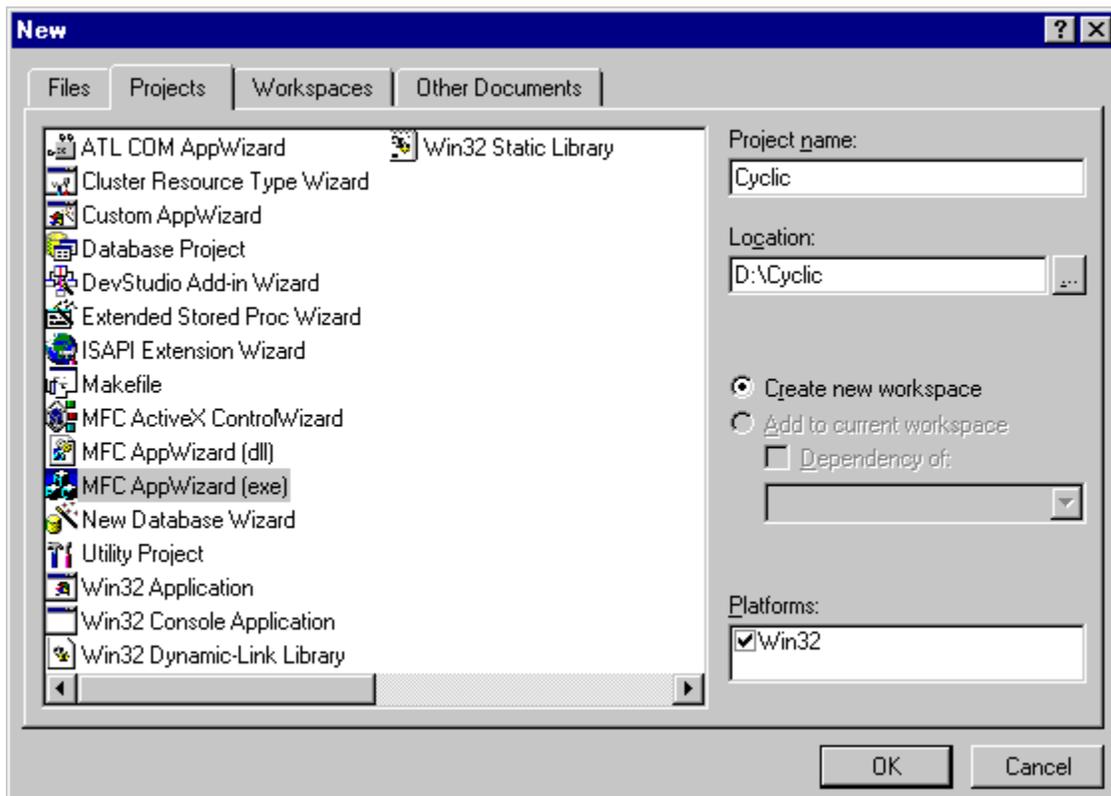
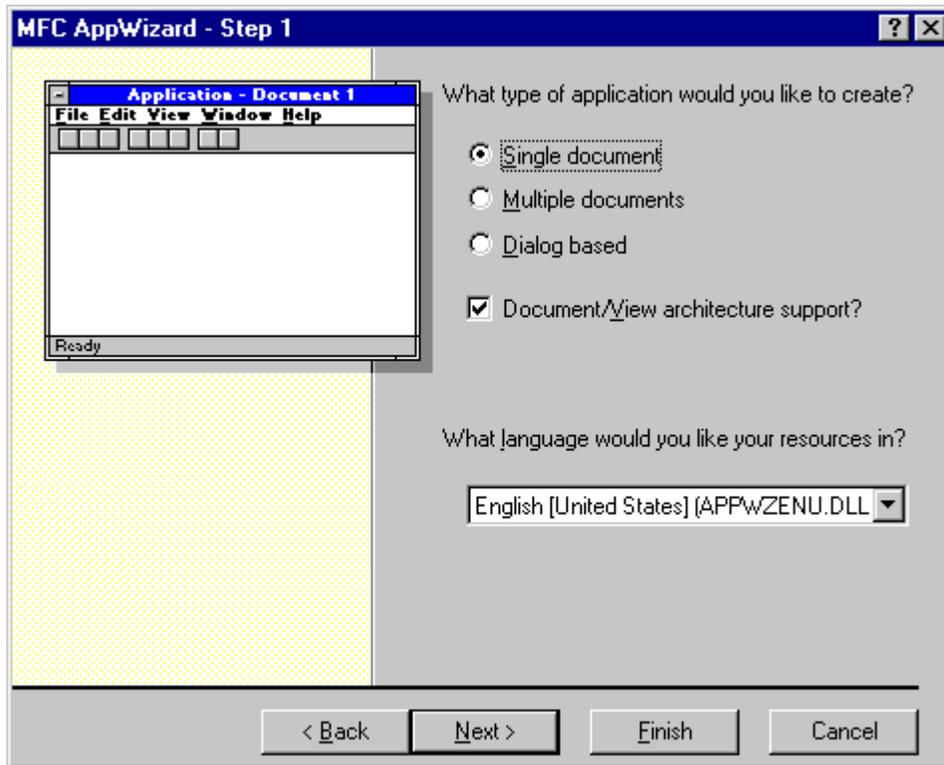


Figure 5 New Dialog Box

4. The next dialog you will see is the MFC AppWizard - Step 1 dialog. In this dialog select Single document, Document/View architecture support, and English. After you have entered this information your dialog should look like the one in Figure 6. After you are finished push **Next**.



**Figure 6** MFC AppWizard - Step 1

5. The next dialog you will see is the MFC AppWizard – Step 2 of 6 dialog. You do not need to make any changes to the options in this dialog. It should look like the one in Figure 7. Push **Next**.

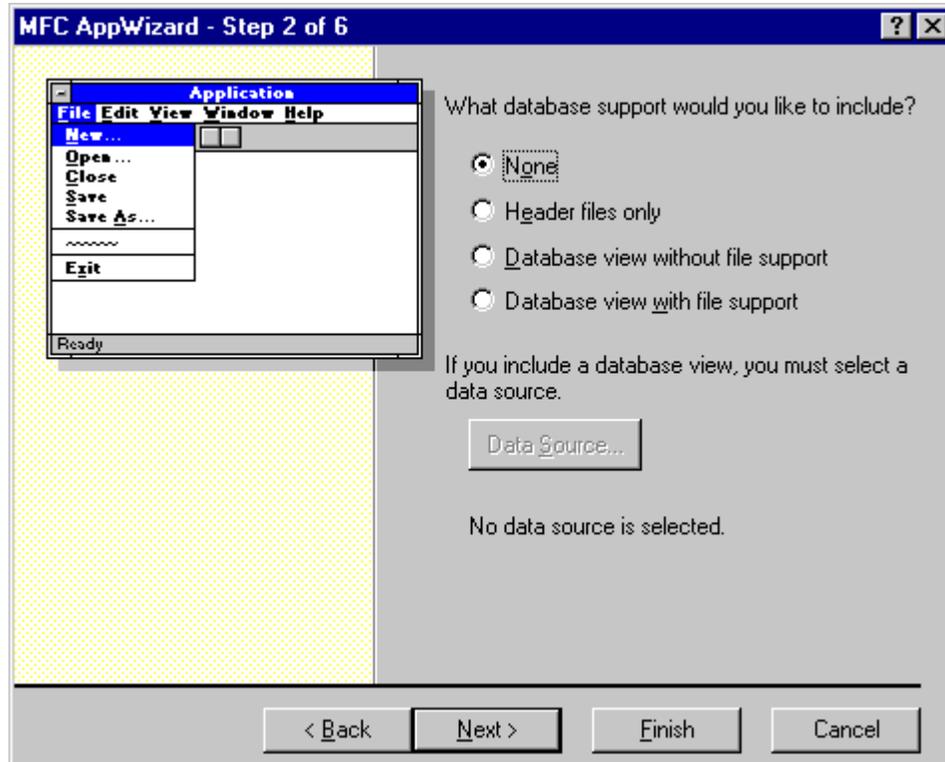


Figure 7 MFC AppWizard – Step 2

- The next dialog you will see is the MFC AppWizard – Step 3 of 6 dialog. You do not need to make any changes to the options in this dialog. It should look like the one in Figure 8. Push **Next**.

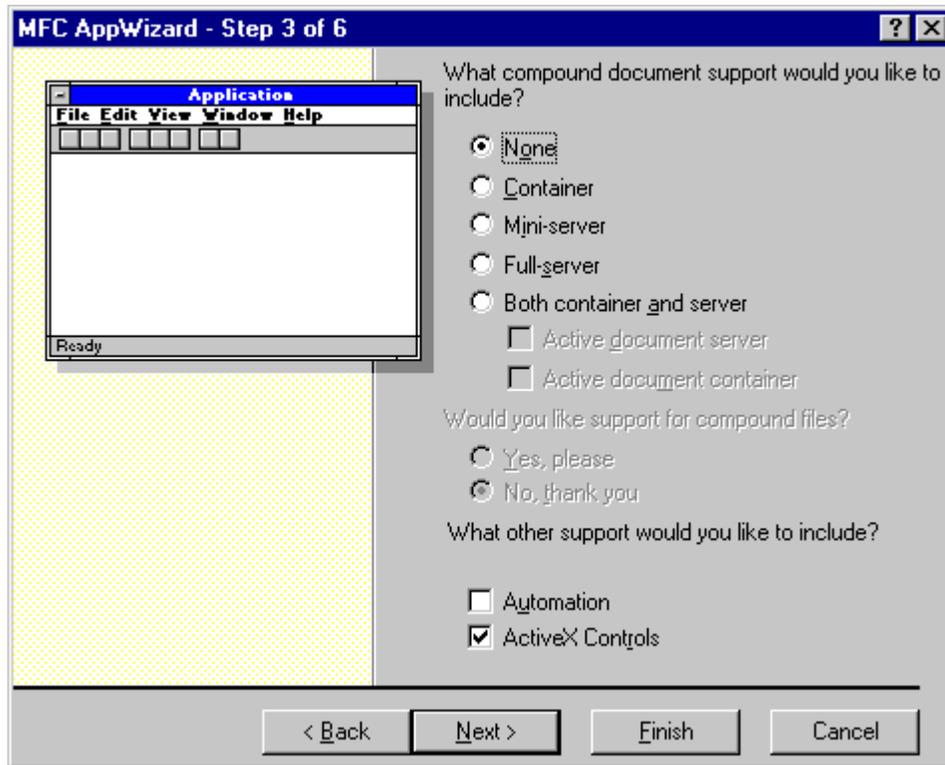


Figure 8 MFC AppWizard – Step 3

7. The next dialog you will see is the MFC AppWizard – Step 4 of 6 dialog. You do not need to make any changes to the options in this dialog. It should look like the one in Figure 9. Push **Next**.

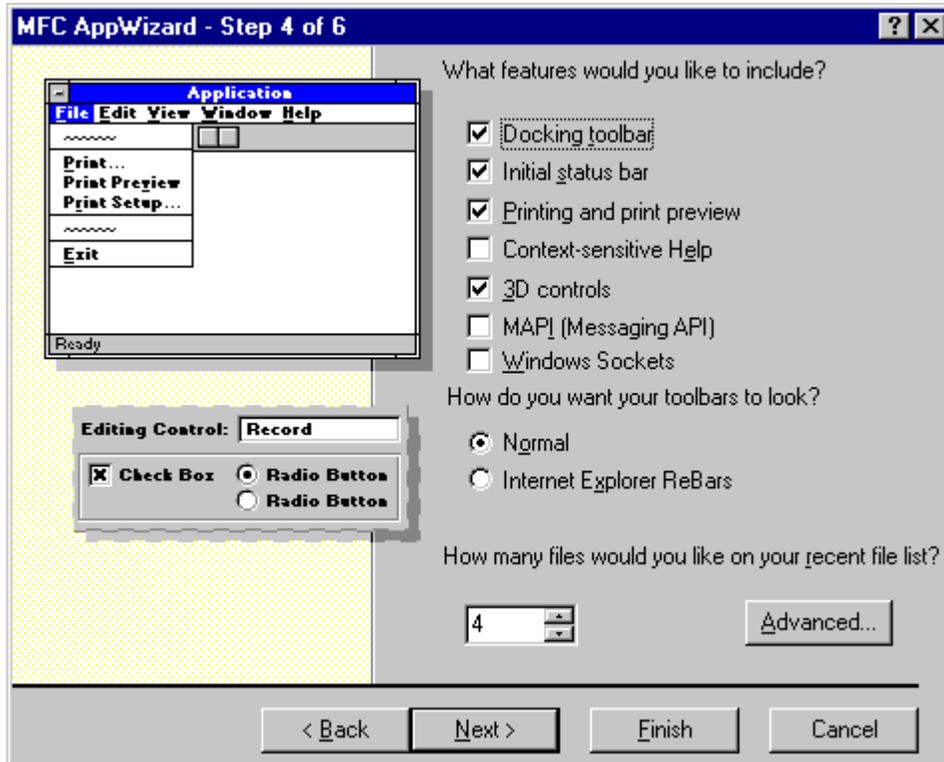


Figure 9 MFC AppWizard – Step 4

8. The next dialog you will see is the MFC AppWizard – Step 5 of 6 dialog. You do not need to make any changes to the options in this dialog. It should look like the one in Figure 10. Push **Next**.

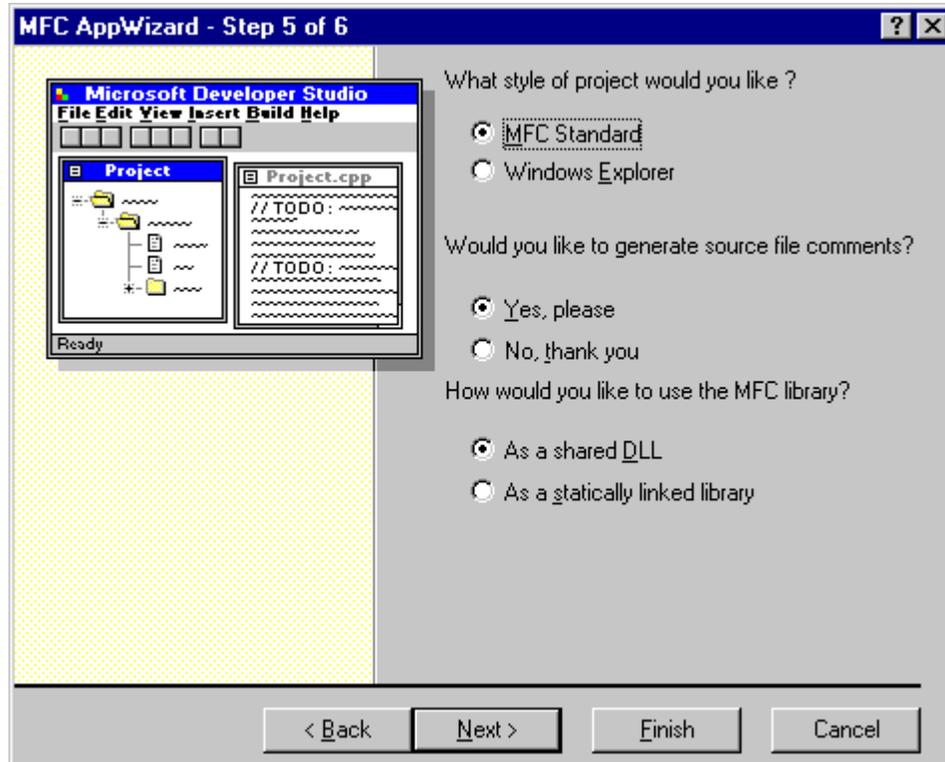


Figure 10 MFC AppWizard – Step 5

9. The next dialog you will see is the MFC AppWizard – Step 6 of 6 dialog. When this dialog appears it will look like the one in Figure 11.

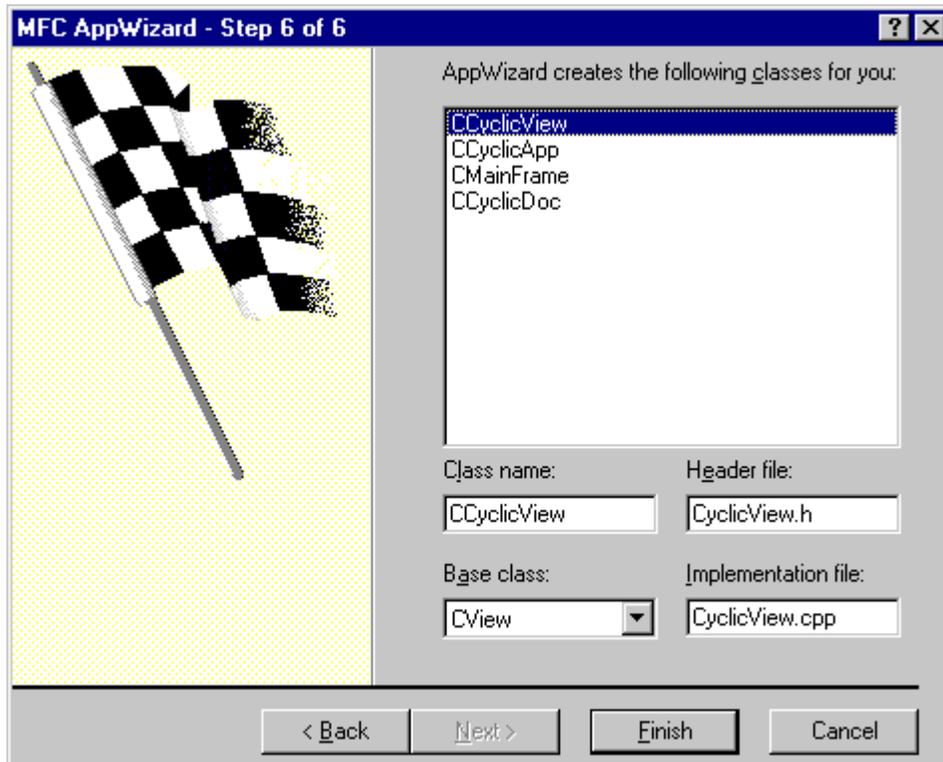


Figure 11 MFC AppWizard – Step 6

- Using the Base class menu change the Base class to **CFormView**. The FormView will allow you to place controls in the main window. This is the last change you need to make in this dialog. Your dialog should look like the one in Figure 12. When you are finished push the **Finish** button.

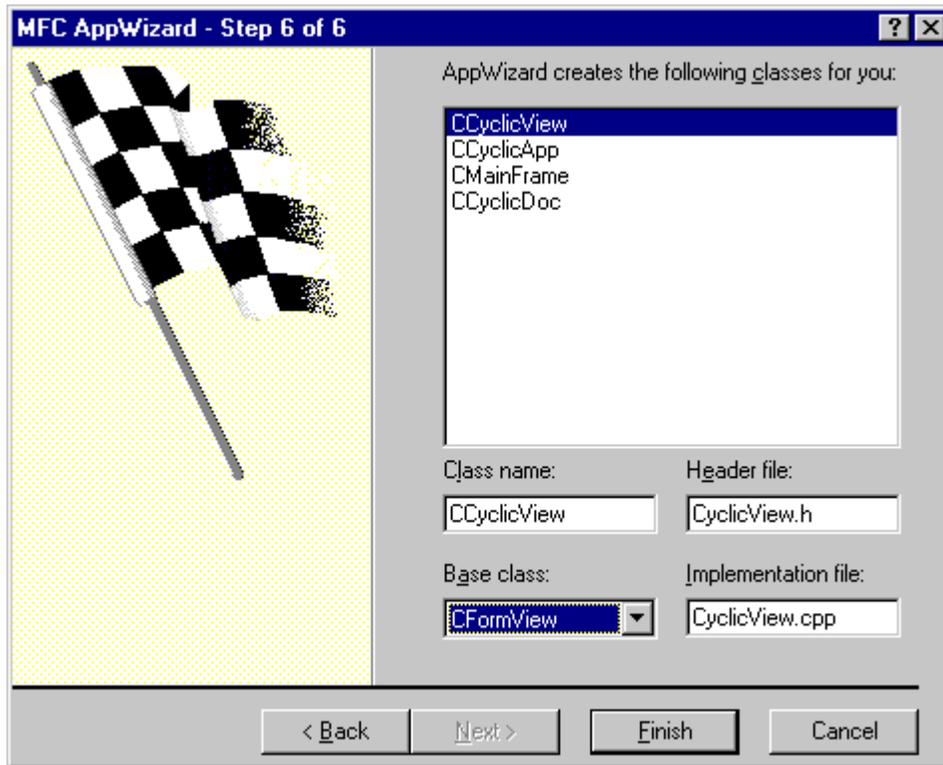
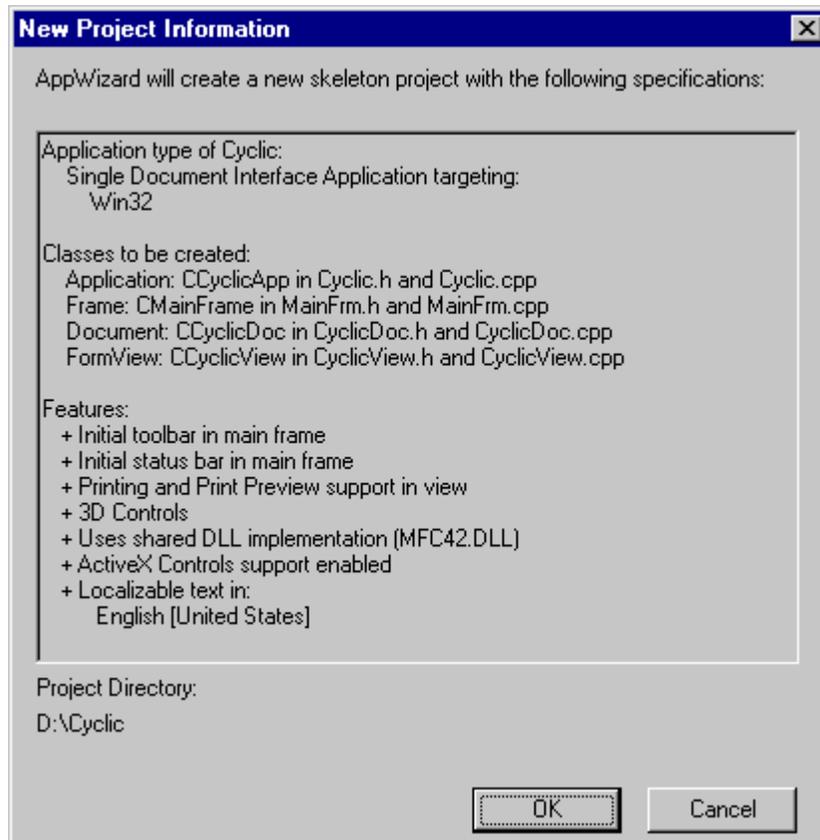


Figure 12 MFC AppWizard - Step 6 Dialog showing Base class CFormView Selection.

11. The last dialog you will see is the New Project Information dialog. This dialog should look like the one in Figure 13. Push the **OK** button and Visual C++ will create the skeleton application files for you.



**Figure 13** New Project Information dialog.

- At this point you should now have a directory called Cyclic that was created in the location you specified. Since Visual C++ has created a complete application you can compile and run the application to see what it looks like so far. Go to the **Build** menu and select **Build Cyclic.exe**. You should see something similar to the following messages in the message area at the bottom of the Visual C++ window.

-----Configuration: Cyclic - Win32 Debug-----

Compiling resources...

Compiling...

StdAfx.cpp

Compiling...

Cyclic.cpp

MainFrm.cpp

CyclicDoc.cpp

CyclicView.cpp

Generating Code...

Linking...

Cyclic.exe - 0 error(s), 0 warning(s)

13. After Visual C++ has finished building the application, go to the **Build** menu and select **Execute Cyclic.exe**. When you do this you should see the cyclic application main window as shown in Figure 14.



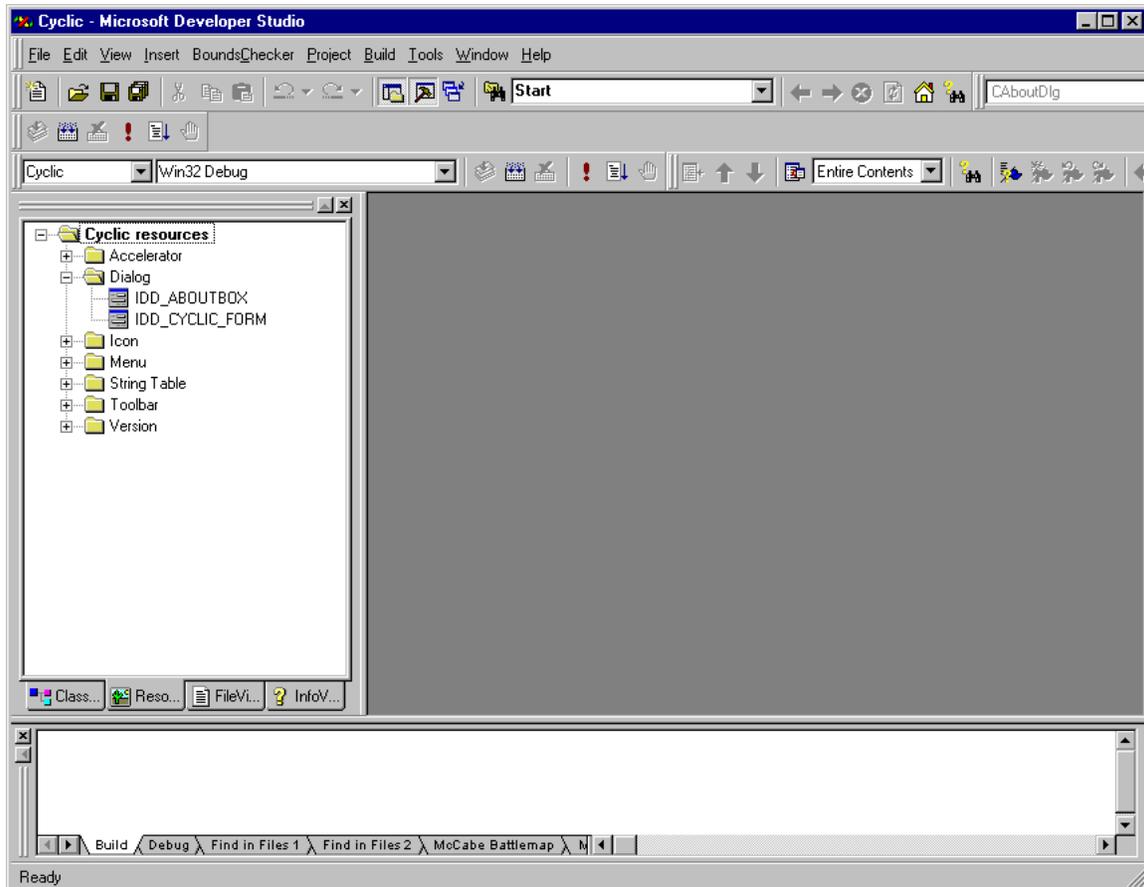
**Figure 14 Cyclic Application Main Window**

14. At this point you can now start adding in your application specific code. Exit the Cyclic application and move on to Part II when you're ready.

## **Part II**

### **Adding the Menus and Controls to the Main Window**

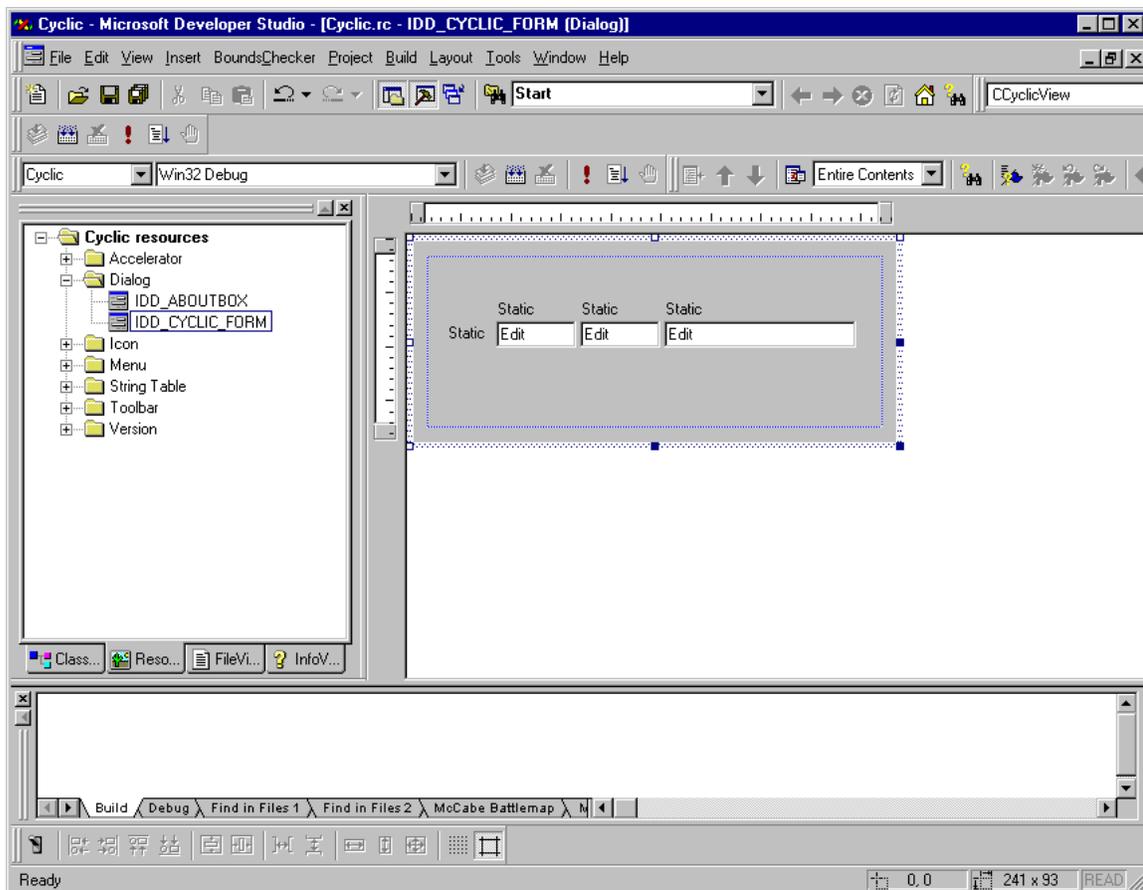
1. In the Visual C++ window choose the Resource Tab and use the tree control to show the list of dialogs. Your Visual C++ window should look like the one in Figure 15.



**Figure 15 Visual C++ Resource Tab**

2. Double click on IDD\_CYCLIC\_FORM. This will open up the form view control that serves as the client area of the main window of the Cyclic application. Perform the following steps:
  - Delete the static text control displaying the text “TODO: Place form controls on this dialog.”.
  - Use the Visual C++ tools to place four static text controls and three edit controls onto the form as shown in Figure 16.

The three edit controls will be used to display the converted integer value of MSID038, the status string for MSID038, and the TReK API Return value for the API call used to retrieve the MSID038 converted value.



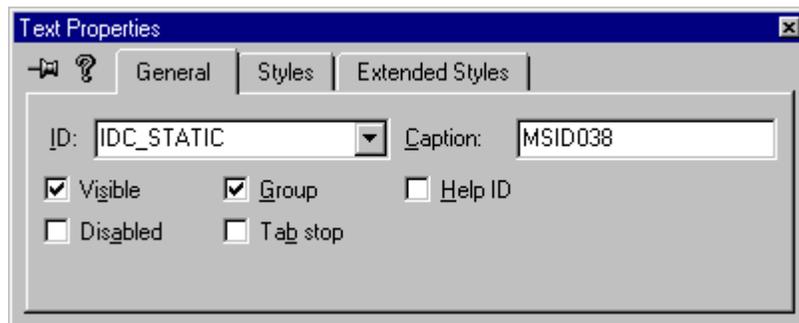
**Figure 16** The IDD\_CYCLIC\_FORM after the controls have been placed on the form.

Note: If you inadvertently close the Controls palette, you can get it back by putting your mouse cursor on the Visual C++ menu bar and right clicking to activate the pop-up menu. Choose Controls on the pop-up menu and the floating Controls palette will reappear.

3. Select the static control on the far left, click the right mouse button, and select properties. The Properties dialog shown in Figure 17 will appear. The word Static is already selected. Type in the text MSID038 to replace the word Static. The dialog should now look like the one in Figure 18. Close the dialog by selecting the X in the upper right hand corner.



**Figure 17 Static Control properties dialog.**



**Figure 18 Static control properties dialog showing MSID038 caption.**

4. Change each of the other Static controls in the same way. The other static controls from left to right should display the words Value, Status, and API Return in that order. After you are finished your controls should look like the ones in Figure 19.

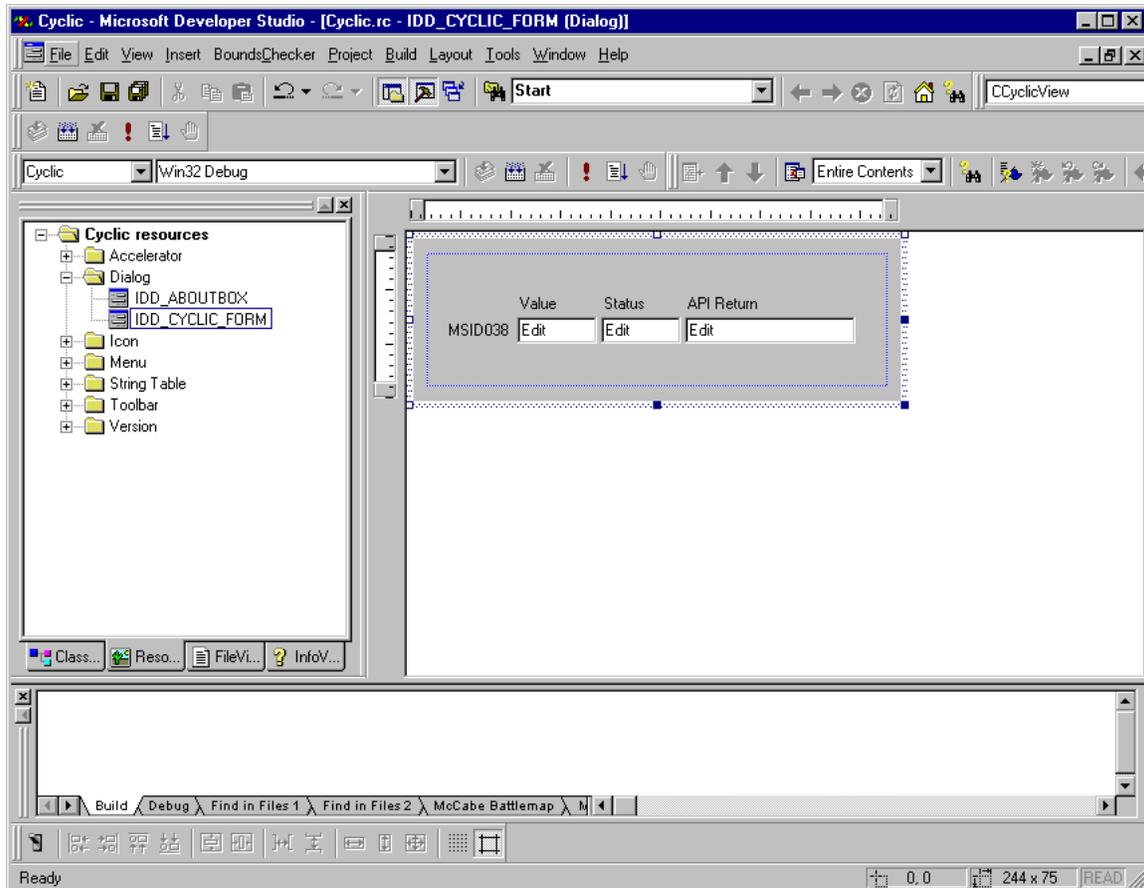


Figure 19 Cyclic Main Window Controls

5. Select the edit control on the far left, click the right mouse button, and select properties. The Properties dialog shown in Figure 20 will appear. Change the ID field so the edit control's ID is IDC\_MSID038\_INT\_VALUE\_EDIT as shown in Figure 21. (Note: You can not see all of the text in the text field in the picture, but make sure you type in the complete name IDC\_MSID038\_INT\_VALUE\_EDIT).



Figure 20 Edit Control Before Control ID Change.

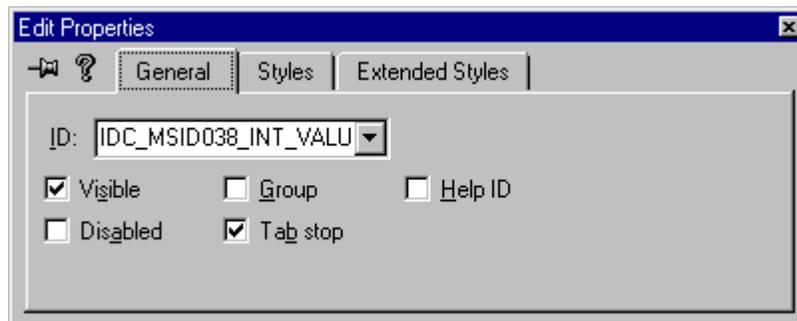


Figure 21 Edit Control After Control ID Change.

6. Change the other two edit control IDs in the same manner. The other two IDs from left to right should be IDC\_MSID038\_INT\_STATUS\_EDIT and IDC\_MSID038\_INT\_API\_EDIT.

- At this point you need to assign member variables to each of the edit controls. Go to the View menu and select **ClassWizard...** In the Class Wizard dialog select the Member Variables tab. Make sure the Class name is CCyclicView. The member variables you are about to create should be part of the View class. The ClassWizard dialog should look like the one in Figure 22.

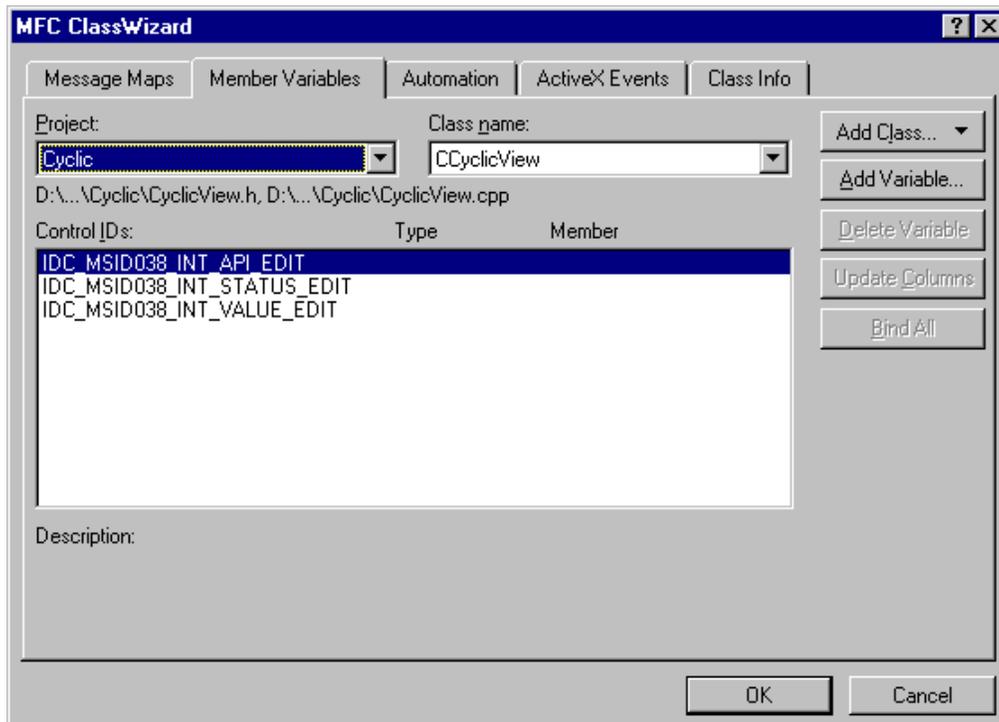
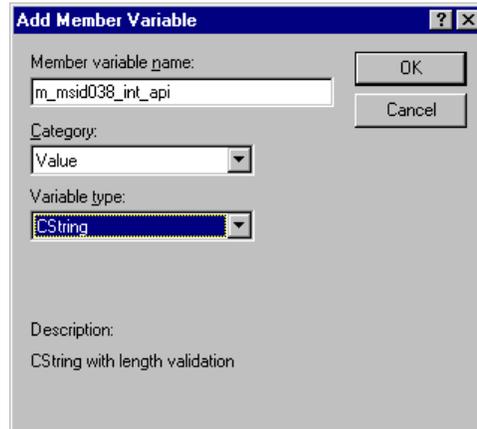


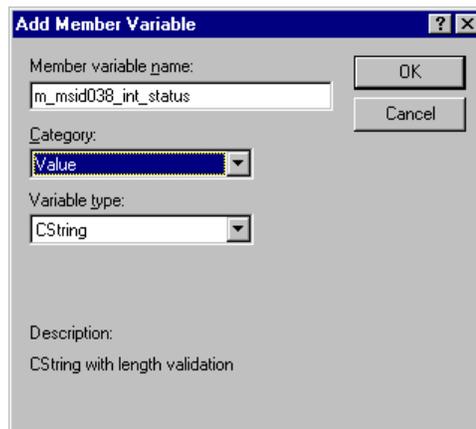
Figure 22 ClassWizard Dialog Before Adding Member Variables

- In the ClassWizard dialog select the IDC\_MSID038\_INT\_API\_EDIT control and then push the Add Variable button. In the Add Variable dialog fill in the information shown in Figure 23 (Member variable name: m\_msid038\_int\_api, Category: Value, Variable type: CString). This member variable will be used to hold the API function call return value (in a character string format). You will learn more about this in Part IV. After you are finished push the **OK** button.



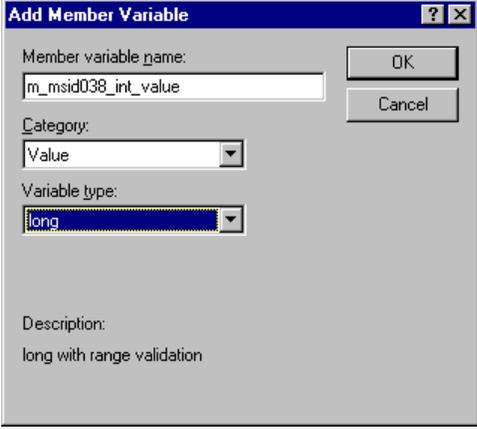
**Figure 23 Add Member Variable for IDC\_MSID038\_INT\_API\_EDIT**

- Choose the IDC\_MSID038\_INT\_STATUS\_EDIT variable and push the **Add Variable** button. Enter the information shown in Figure 24 (Member variable name: m\_msid038\_int\_status, Category: Value, Variable type: CString). This member variable will be used to hold the status string associated with MSID038. This is returned in the API function call. You will learn more about this in Part IV. Push the **OK** button.



**Figure 24 Add Member Variable for IDC\_MSID038\_INT\_STATUS\_EDIT**

10. Choose the IDC\_MSID038\_INT\_VALUE\_EDIT variable and push the **Add Variable** button. Enter the information shown in Figure 25 (Member Variable Name: m\_msid038\_int\_value, Category: Value, Variable Type: long). This member variable will be used to hold the value of MSID038 that is retrieved using the API function call. You will learn more about this in Part IV. Push the **OK** button.

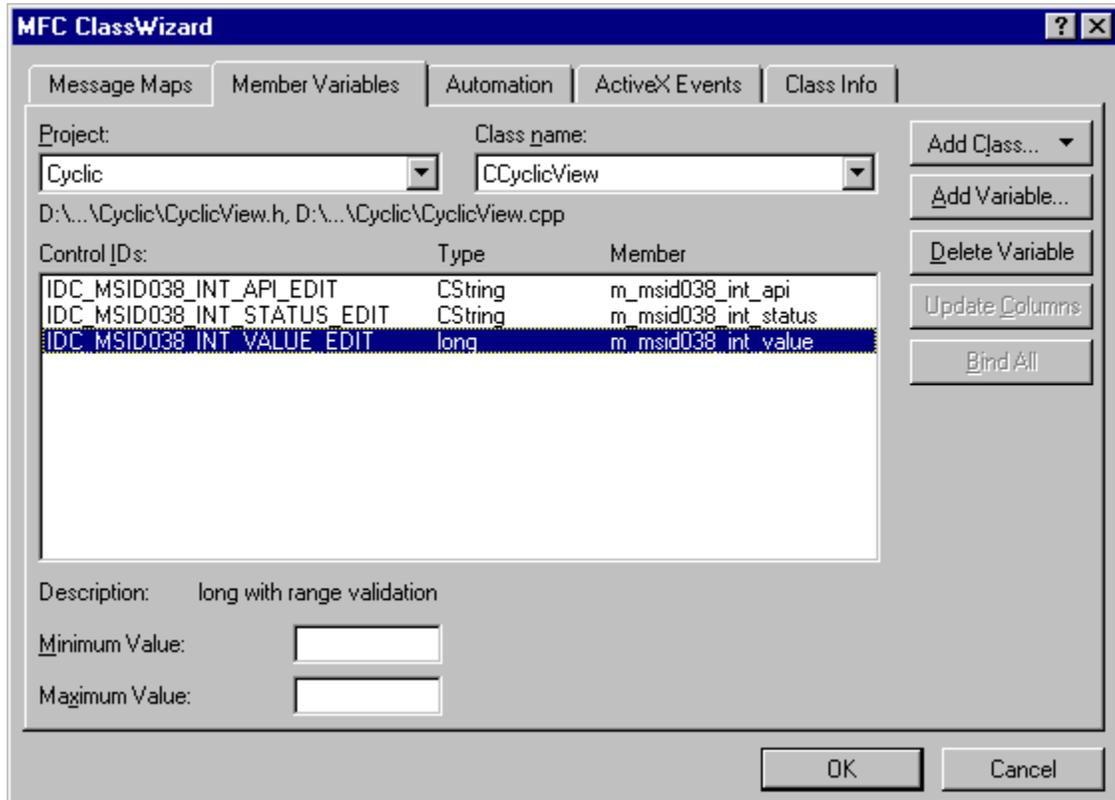


The image shows a dialog box titled "Add Member Variable". It has a standard Windows-style title bar with a question mark icon and a close button. The dialog contains the following fields and controls:

- Member variable name:** A text input field containing the text "m\_msid038\_int\_value".
- Category:** A dropdown menu currently showing "Value".
- Variable type:** A dropdown menu currently showing "long".
- Description:** A text area containing the text "long with range validation".
- Buttons:** "OK" and "Cancel" buttons are located on the right side of the dialog.

**Figure 25 Add Member Variable for IDC\_MSID038\_INT\_VALUE\_EDIT**

11. After you have added all the member variables, the Class Wizard dialog should look like the one in Figure 26.



**Figure 26 Class Wizard after adding Member Variables**

12. In the ClassWizard dialog choose the **Message Maps** tab. The dialog should now look like the one in Figure 27.

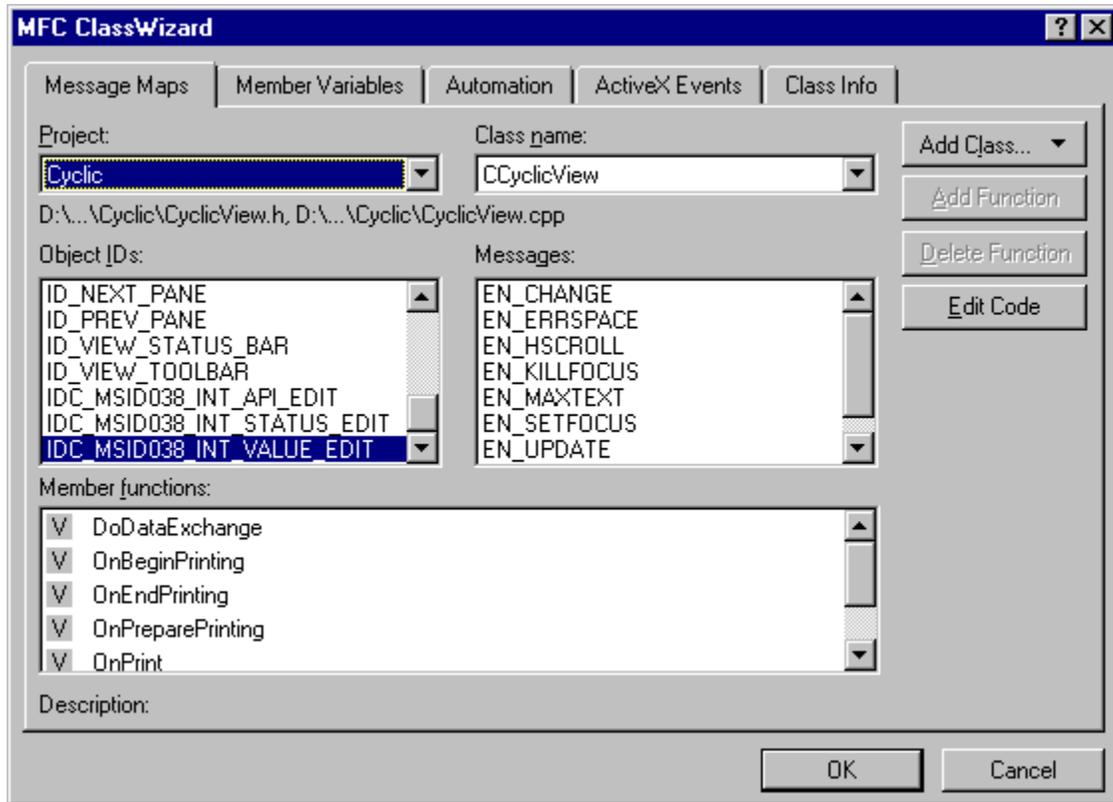


Figure 27 ClassWizard Dialog with the Message Maps tab selected.

- In the Object IDs list scroll to the top of the list and select **CCyclicView**. In the Messages list on the right select the **OnUpdate** message. Once you have **OnUpdate** selected as shown in Figure 28, push the **Add Function** button. After you have completed this step push the **OK** button. Save your work by selecting **Save All** from the **File** menu.

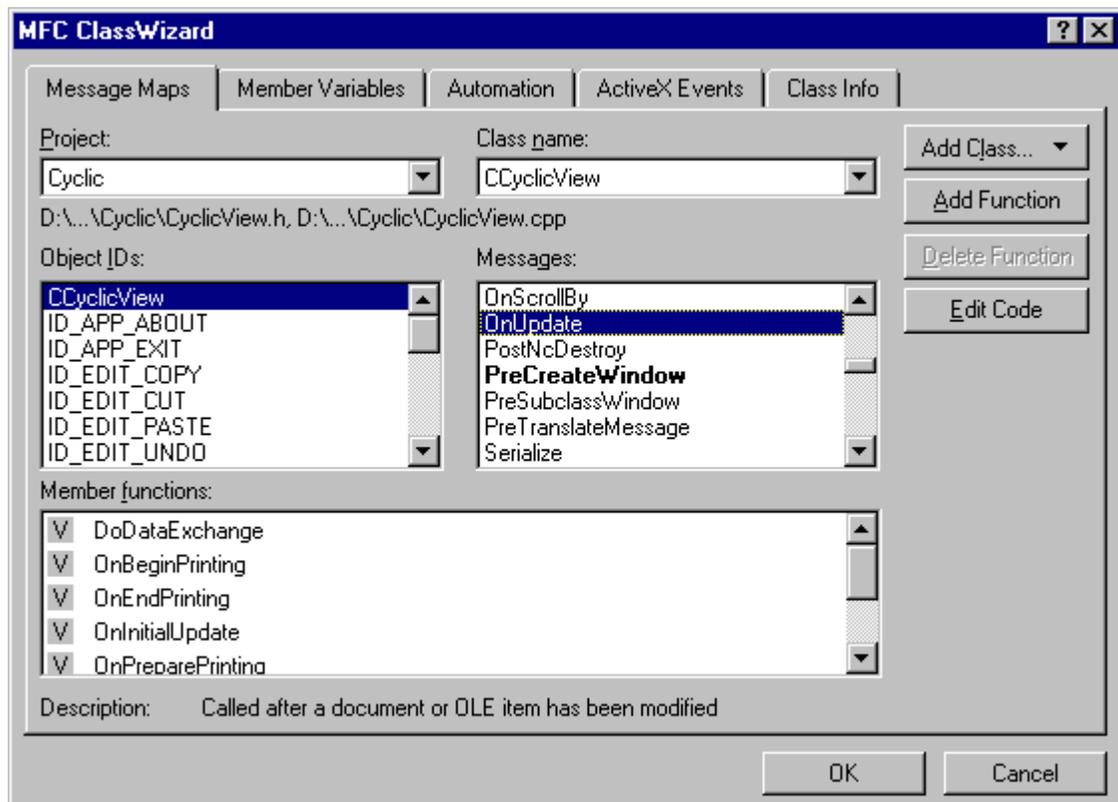
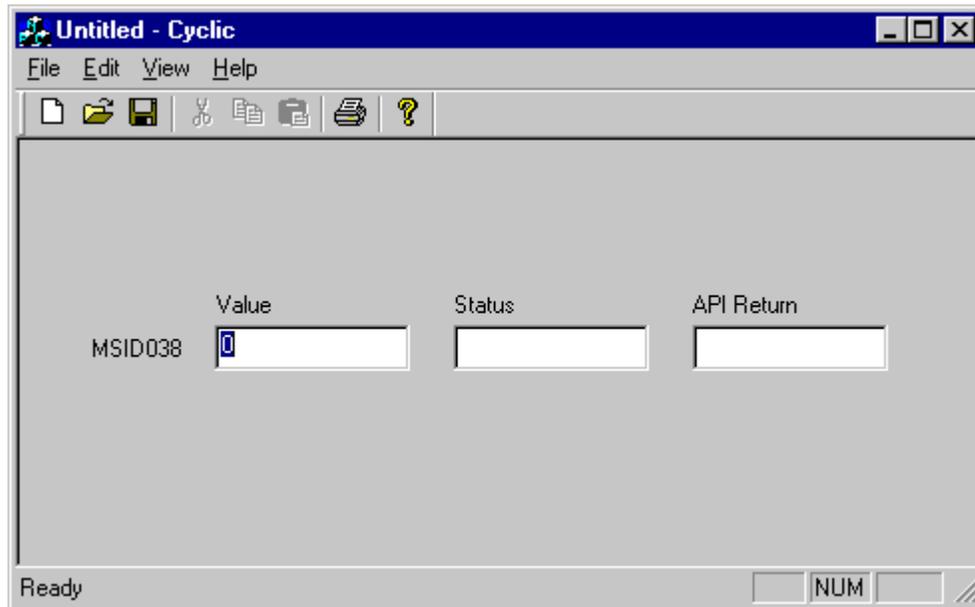


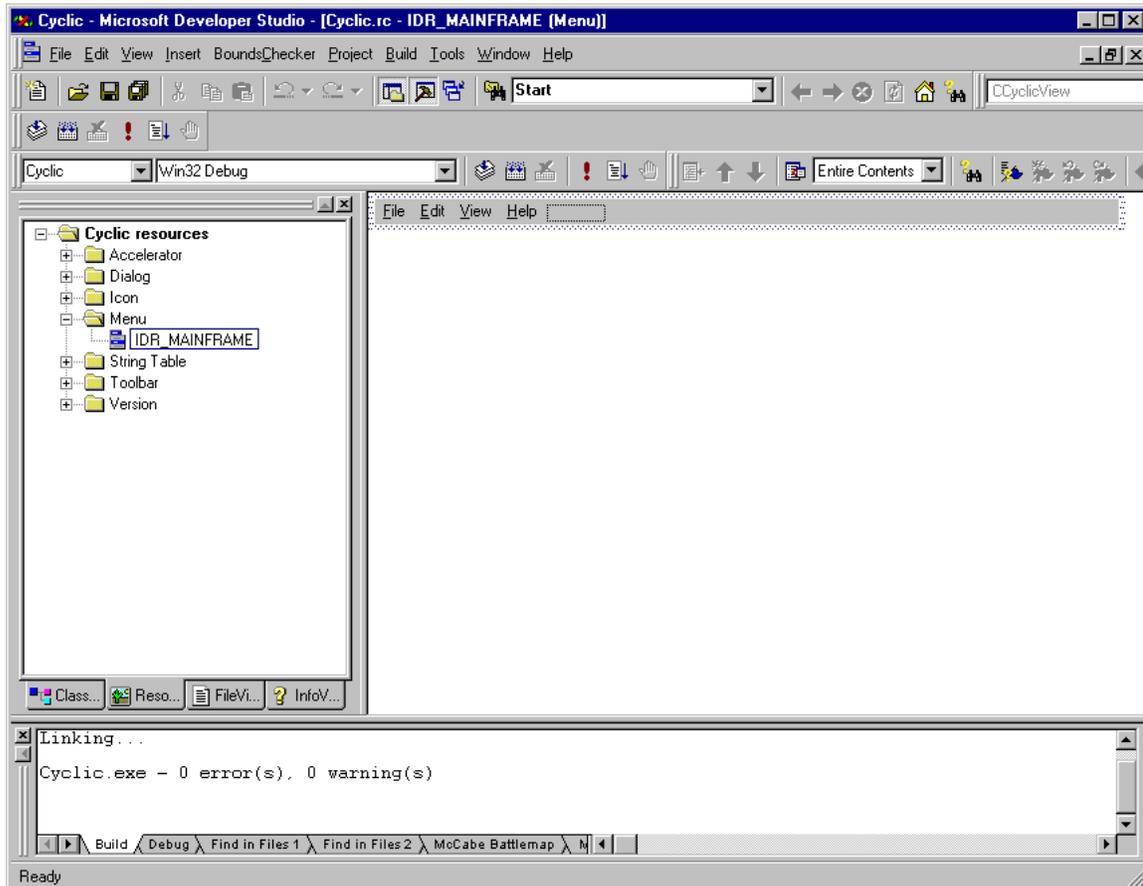
Figure 28 ClassWizard OnUpdate Message

14. Rebuild and Execute the application just to make sure everything is okay at this point. You should now see a window like the one shown in Figure 29. After you have finished looking at it exit the Cyclic application.



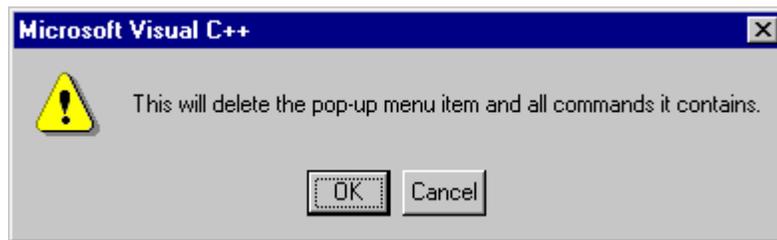
**Figure 29** Cyclic main window with static and edit controls.

- Now we need to fix up the Cyclic application's menu bar. This will require deleting a few items and adding a few. Select the Resource Tab in Visual C++. Under Cyclic Resources, select Menu and then double click on the IDR\_MAINFRAME item. This will open up the Menu Bar resource so you can edit it. Your Visual C++ window should look similar to the one in Figure 30.



**Figure 30 Visual C++ Menu Resource Editor**

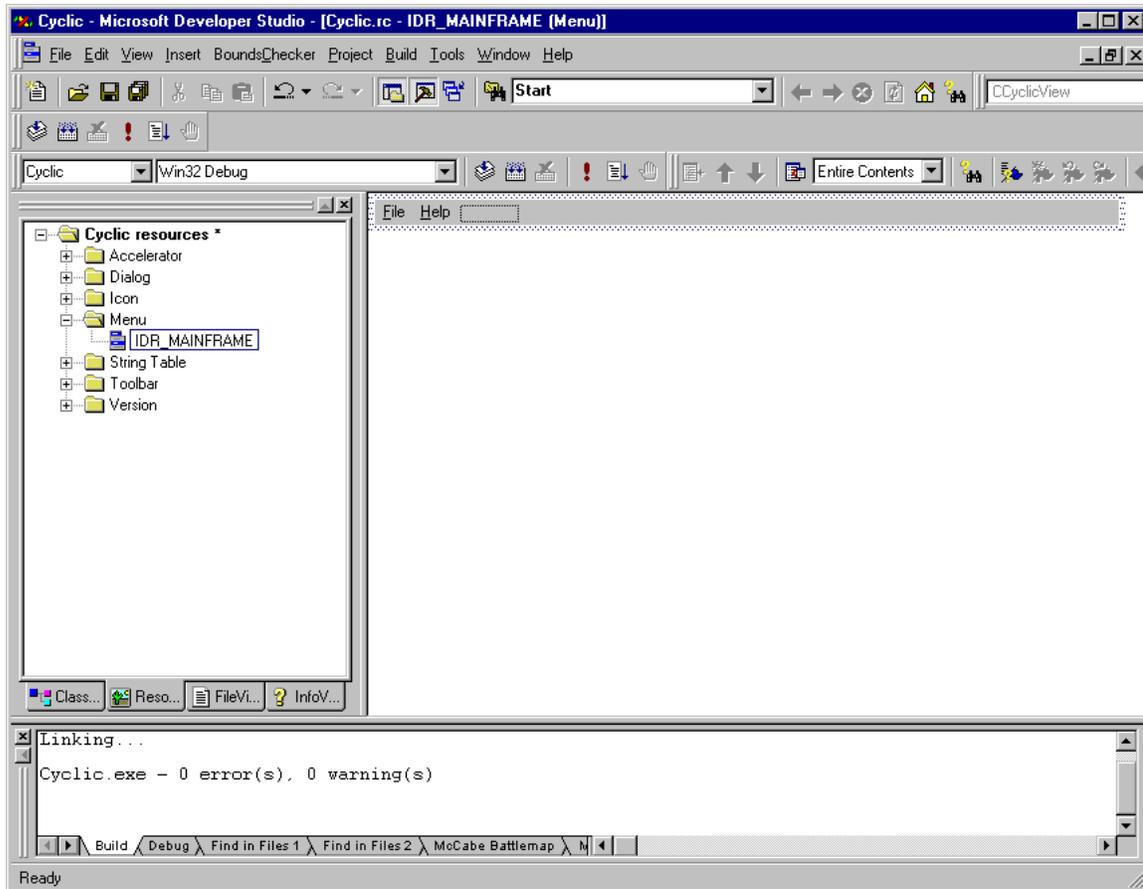
16. The first order of business is to delete the items that are not needed in the Cyclic application (which is most of them). Select the File menu so the menu is posted (appears). Select and delete all of the items on the File menu except the Exit item.
  
17. Delete the entire Edit menu by selecting the Edit menu on the Menu Bar you are modifying and push the delete key. Visual C++ will show you warning in Figure 31. Just push **OK**.



**Figure 31 Edit Menu Deletion Warning.**

18. Delete the View Menu in the same manner. You will get another warning. Just push **OK**.

19. Now it's time to create a new menu called Update. Select the empty square frame at the end of the menu bar. This empty frame can be seen in Figure 32. It is located to the right of the help menu.



**Figure 32** Menu Resource Editor showing Empty Menu Frame

20. Select the empty menu frame and drag it so it is between the File menu and the Help Menu. While the Frame is still selected type the word **Update**. When you begin typing the Properties dialog will appear. When you are done typing the word **Update** close the Properties dialog. Your menu bar should look like the one in Figure 34.

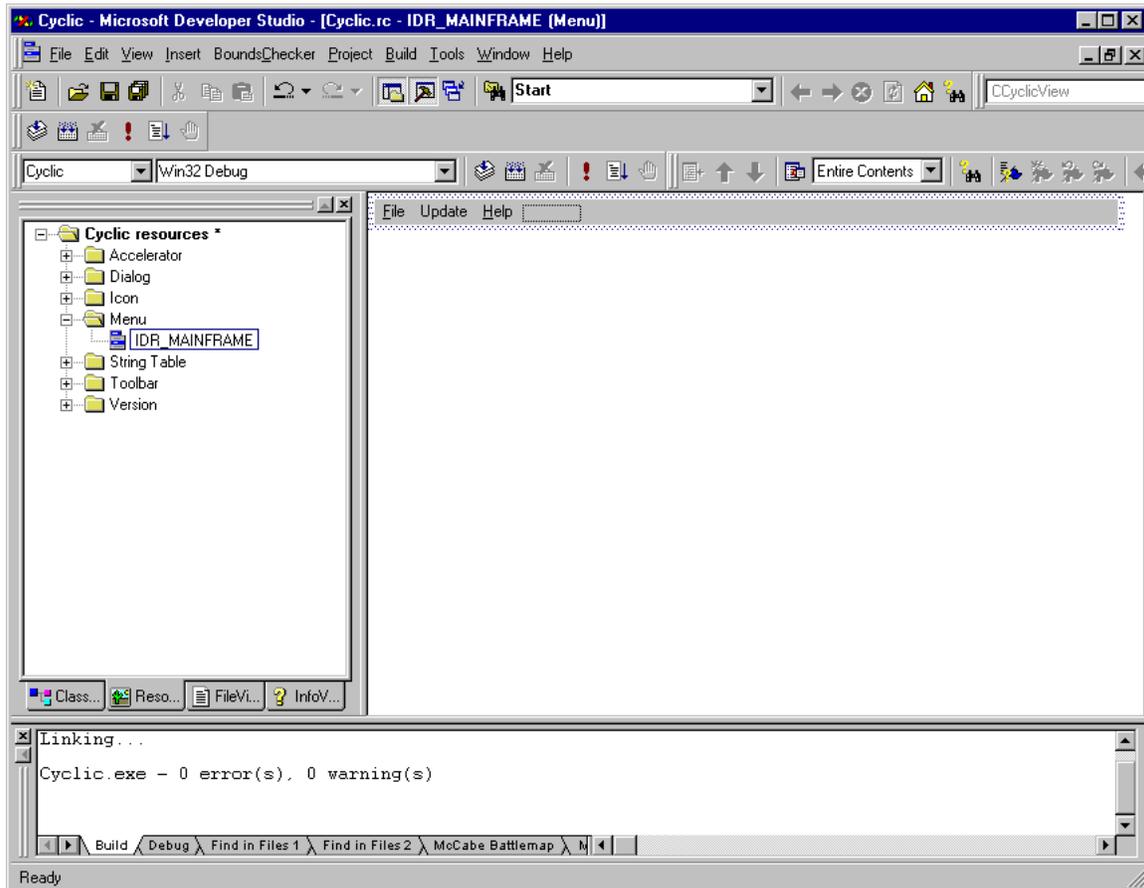


Figure 34 Cyclic Application with Update Menu

21. Now you need to add two new menu items to the Update menu. Select the Update menu so it is posted. Select the first blank frame. Click on the right mouse button and select Properties. In the Properties dialog type in **Start** for the Caption and **IDM\_UPDATE\_START** for the ID. Your dialog should look like the one in Figure 35. Close the Properties dialog.



Figure 35 Update Menu with Start Item

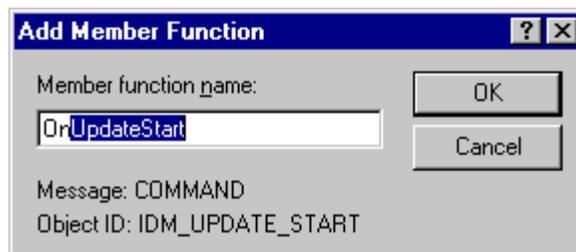
22. Now add the second menu item to the Update menu. Select the Update menu so it is posted. Select the first blank frame. Click on the right mouse button and select Properties. In the Properties dialog type in **Stop** for the Caption and **IDM\_UPDATE\_STOP** for the ID. Your dialog should look like the one in Figure 36. Close the Properties dialog.



Figure 36 Update Menu with Stop Item

23. Now the Update menu is complete. Choose **Save All** from the **File** menu to save your work.
24. In order for these menu items to work, you need to add a message handler for each one. Go to the View menu and choose **ClassWizard...** In the **ClassWizard** dialog select the **Message Maps** tab. Perform the following steps:
- In the Class name list make sure **CCyclicView** is selected.
  - In the Object IDs list select **IDM\_UPDATE\_START**.
  - In the Messages list on the right, select **COMMAND**.
  - Push the **Add Function...** button.

The Add Member function message dialog will appear as shown in Figure 37. Don't make any changes, just push OK.

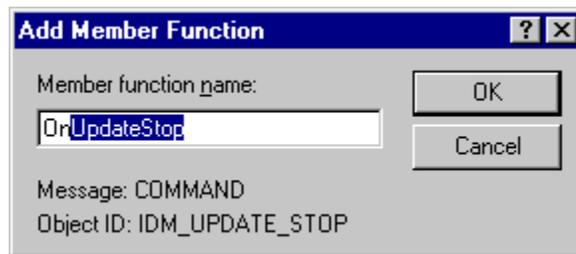


**Figure 37** Add Member Function message dialog.

25. Now you need to perform the same steps for the Stop menu item. Perform the following steps:

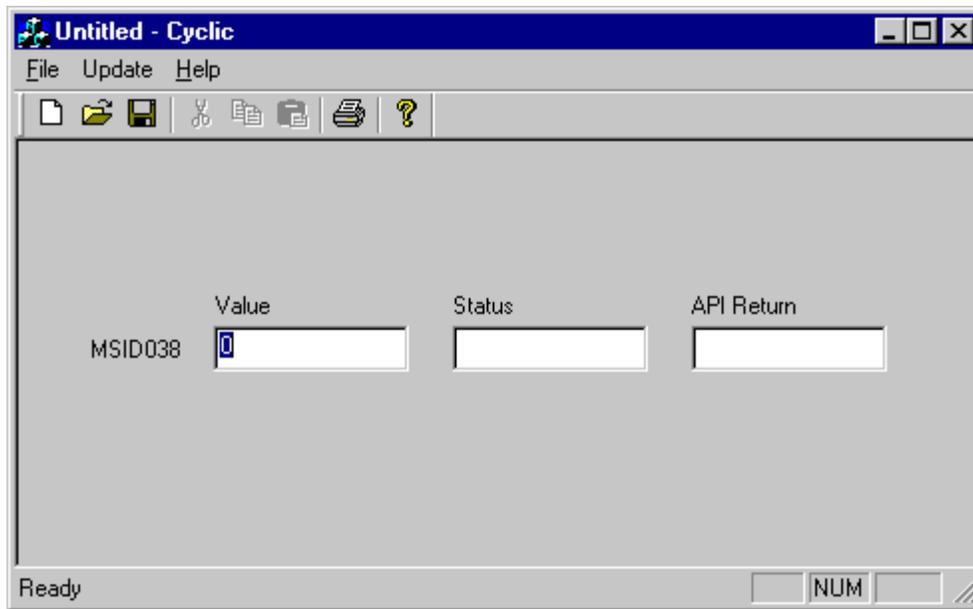
- In the Class name list make sure **CCyclicView** is selected.
- In the Object IDs list select **IDM\_UPDATE\_STOP**.
- In the Messages list on the right, select **COMMAND**.
- Push the **Add Function...** button.

The Add Member function message dialog will appear as shown in Figure 38. Don't make any changes, just push OK.



**Figure 38 Add Member Function dialog.**

26. Push the OK button in the ClassWizard Dialog. Choose **Save All** from the **File** menu to save your work.
  
27. Rebuild and Execute the application just to make sure everything is okay at this point. You should now see a window like the one shown in Figure 39. After you have finished looking at it exit the cyclic application.



**Figure 39 Cyclic Application with Update Menu**

28. This concludes Part II. You're now ready to move on to Part III.

## **Part III**

### **Adding The Timer Thread**

1. Now the real fun begins. It's now time to start adding in the code that will make the application do something interesting. This code will be added in both Part III and Part IV. The objective of Part III is to add the user interface thread. Remember that this thread makes it possible for your application to appear to be doing two things at one time. To get started you need to add four new files to the Cyclic project. In order to save time these files have been created for you and can be found in the TReK installation directory under templates. Perform the following steps to add these files to your project:

- Go to the TReK installation directory. Look in the templates directory and copy the following files into your Cyclic project directory:
  - UpdateInfo.cpp
  - UpdateInfo.h
  - UpdateThread.cpp
  - UpdateThread.h
  
- In Visual C++, go to the **Project** menu, select the **Add To Project** cascade menu, and then select **Files....** The **Insert Files Into Project** dialog will appear. In the **Insert Files Into Project** dialog select all four files as shown in Figure 40 and then push the **OK** button. (Hint: Multiple files can be selected by holding down the Ctrl key during the selection.)

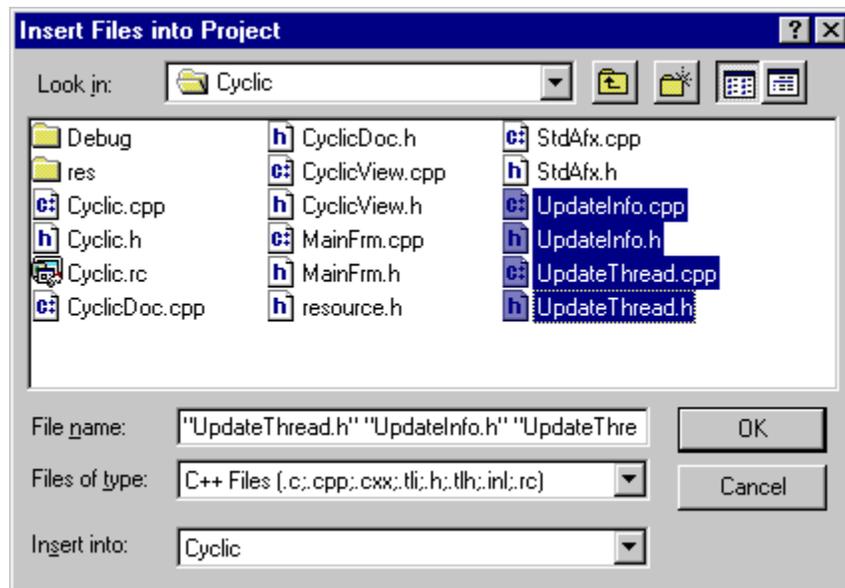


Figure 40 Insert Files into Project Dialog

2. In Visual C++ click on the **FileView** tab to see the list of Files that are in the project. You should now see the four files you just added as shown in Figure 41.

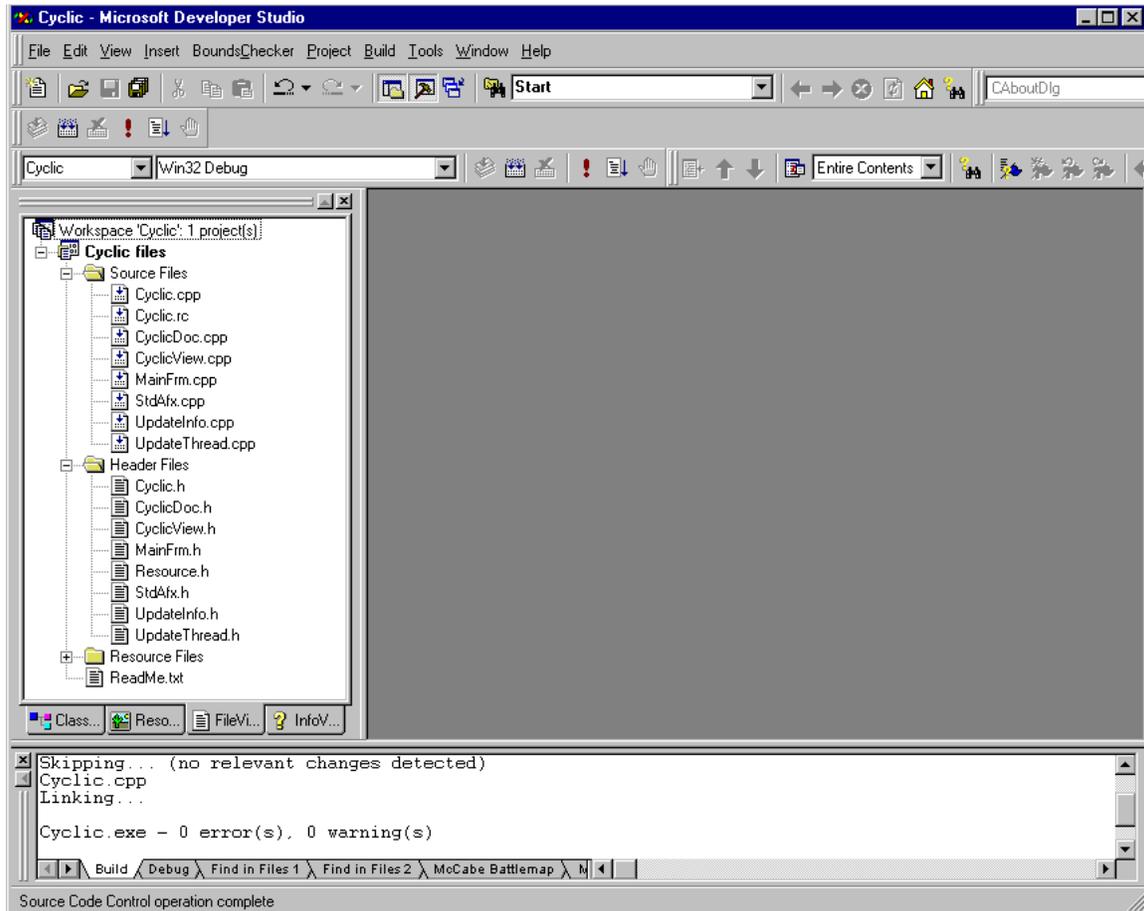


Figure 41 Cyclic project showing the project files list.

3. The four files that you added will be used to create the user interface thread discussed in the Introduction. By having a separate thread you can still access the application menu items while the information in the main window is updated every second. There is a little bit of set up that needs to be done in the **CyclicView.cpp** file in order to use these files. Go to the **FileView** tab in the Visual C++ window and double click on the **CyclicView.cpp** file in order to open it. Go to the top of the file and add the following lines of code after the Debug defines.

```
#include "UpdateThread.h"
#include "UpdateInfo.h"

#define WM_MYMESSAGE (WM_USER + 100)
```

4. The top of your **CyclicView.cpp** file should look like the following segment of code. The two include statements include the header files so you can use the functions in the UpdateThread and UpdateInfo files. The #define statement defines a user-defined message that you will be using later.

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#include "UpdateThread.h"
#include "UpdateInfo.h"

#define WM_MYMESSAGE (WM_USER + 100)
```

5. Go to the **File** menu and select **Save All** to save your work.

6. During the next few steps you will be adding a user defined message. This is the message that the timer thread will post to the View class once every second. In this message handler the View class calls the Document class to tell it to go get new data. In the **CCyclicView.cpp** file, scroll down until you find the Message Map section. This is what you are looking for:

```

////////////////////////////////////
/////
// CCyclicView

IMPLEMENT_DYNCREATE(CCyclicView, CFormView)

BEGIN_MESSAGE_MAP(CCyclicView, CFormView)
   //{{AFX_MSG_MAP(CCyclicView)
    ON_COMMAND(IDM_UPDATE_START, OnUpdateStart)
    ON_COMMAND(IDM_UPDATE_STOP, OnUpdateStop)
   //}}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CFormView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CFormView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CFormView::OnFilePrintPreview)
END_MESSAGE_MAP()

```

7. Inside the `BEGIN_MESSAGE_MAP(CCyclicView, CFormView)` line and the `END_MESSAGE_MAP()` line add the following line of code after the `//{{AFX_MSG_MAP(CCyclicView)` line:

```
ON_MESSAGE(WM_MYMESSAGE, OnMyMessage)
```

Now the Message Map section should look like the one below.

```

////////////////////////////////////
/////
// CCyclicView

IMPLEMENT_DYNCREATE(CCyclicView, CFormView)

BEGIN_MESSAGE_MAP(CCyclicView, CFormView)
   //{{AFX_MSG_MAP(CCyclicView)
    ON_MESSAGE(WM_MYMESSAGE, OnMyMessage)
    ON_COMMAND(IDM_UPDATE_START, OnUpdateStart)
    ON_COMMAND(IDM_UPDATE_STOP, OnUpdateStop)
   //}}AFX_MSG_MAP
    // Standard printing commands

```

```

    ON_COMMAND(ID_FILE_PRINT, CFormView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CFormView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CFormView::OnFilePrintPreview)
END_MESSAGE_MAP()

```

8. Go to the **File** menu and select **Save All** to save your work.
  
9. In the Files list double click on the **CyclicView.h** file to open it. In the CyclicView.h file scroll down until you see the General Message Map functions section. This is what you're looking for:

```

protected:

// Generated message map functions
protected:
   //{{AFX_MSG(CCyclicView)
    afx_msg void OnUpdateStart();
    afx_msg void OnUpdateStop();
   //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

```

10. Add the following line of code after the `//{{AFX_MSG(CCyclicView)` line.

```
afx_msg LRESULT OnMyMessage(WPARAM wParam, LPARAM lParam);
```

The message map function section should now look like the following segment of code:

```

protected:

// Generated message map functions
protected:
   //{{AFX_MSG(CCyclicView)
    afx_msg LRESULT OnMyMessage(WPARAM wParam, LPARAM lParam);
    afx_msg void OnUpdateStart();
    afx_msg void OnUpdateStop();
   //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

```

11. Go to the **File** menu and select **Save All** to save your work.

12. Now go back to the **CyclicView.cpp** file and add the following message handler.

This is the message handler that will get triggered when the timer thread posts a message to the View class. This message handler can be added anywhere in the file but it's convenient to put it between the GetDocument() function definition and the CCyclicView message handlers section. In this message handler the View class calls the Document class to tell it to go get new data. The View class also tells the Document class to update all views. When the Document class calls UpdateAllViews, the OnUpdate message will be sent to the View Class. As you will see later on in Part IV, the OnUpdate message handler in the View class gets the new data from the Document class and places it in the Cyclic main window.

```
LRESULT CCyclicView::OnMyMessage(WPARAM wParam, LPARAM lParam)
{
    CCyclicDoc *pDoc = GetDocument();

    pDoc->GetNewData();

    pDoc->UpdateAllViews(NULL, 1, NULL);

    return 0;
}
```

Here's what the final result should look like when the message handler is inserted between the CCyclicView::GetDocument() message handler and the CCyclicView message handlers sections.

```
CCyclicDoc* CCyclicView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CCyclicDoc)));
    return (CCyclicDoc*)m_pDocument;
}
#endif // _DEBUG

LRESULT CCyclicView::OnMyMessage(WPARAM wParam, LPARAM lParam)
{
    CCyclicDoc *pDoc = GetDocument();

    pDoc->GetNewData();

    pDoc->UpdateAllViews(NULL, 1, NULL);

    return 0;
}
```

```

}

////////////////////////////////////
/////
// CCyclicView message handlers

```

13. At this point a few additions need to be made to the document class. In the Files list double click on the **CyclicDoc.h** file to open it. Add `#include "UpdateInfo.h"` and `#include "UpdateThread.h"` right before the document class definition as shown below.

```

    #if _MSC_VER >= 1000
    #pragma once
    #endif // _MSC_VER >= 1000

    #include "UpdateInfo.h"
    #include "UpdateThread.h"

    class CCyclicDoc : public CDocument

```

14. In the **CyclicDoc.h** file find the public Attributes section and add the following member variables:

```

    // Attributes
    public:
    int number;
    HANDLE thread_handle;
    int doc_number;

    int data_mode;
    long msid038_con_token[3];

    // Variables for Individual Parameters
    long msid038_int_value;
    CString msid038_int_status;
    char msid038_int_api[70];

    CupdateThread *thread_ptr;

```

15. In the **CyclicDoc.h** file move to the public Operations section and add the following member functions:

```
// Operations
public:
void StartUpdate(CUpdateInfo *info_ptr);
void StopUpdate();
void GetNewData();
```

16. Now you need to add the corresponding information to the source file. In the Files list double click on the **CyclicDoc.cpp** file to open it. At the top of the file, after the `#ifdef _DEBUG` statements add the include statements shown below so the segment of code looks like the following:

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#include "UpdateThread.h"
#include "UpdateInfo.h"
```

17. In the **CyclicDoc.cpp** file locate the `CCyclicDoc` constructor and update it so it matches the following:

```
CCyclicDoc::CCyclicDoc()
{
    // TODO: add one-time construction code here
    number = 0;
    doc_number = 500;
    msid038_con_token[0] = 0;
    msid038_con_token[1] = 0;
    msid038_con_token[2] = 0;
}
```

18. In the **CyclicDoc.h** file you added three function prototypes `GetNewData()`, `StopUpdate()` and `StartUpdate(CUpdateInfo *info_ptr)`. Now you need to add the function definitions to the **CyclicDoc.cpp** file. Add the following functions after the `CCyclicDoc` diagnostics section. The actual code for these functions will be filled in during Part 4.

```

////////////////////////////////////
// CCyclicDoc User Defined Functions
////////////////////////////////////
void CCyclicDoc::StartUpdate(CUpdateInfo *info_ptr)
{
}

void CCyclicDoc::StopUpdate()
{
}

void CCyclicDoc::GetNewData()
{
}

```

19. Go to the **File** menu and select **Save All** to save your work.

20. You're just about ready to move to Part IV to perform the last steps. However, before you move on to Part IV rebuild and execute the application just to make sure everything is okay. If everything compiles and executes as it should move to Part IV. If you run into any problems remember that you can look at the finished version of this application which is located in the the TReK installation directory under `examples/VisualC++/Cyclic`.

## **Part IV**

# **Adding the Call to the TReK Application Programming Interface Library**

1. Now for the last few steps. It seems like it has taken forever to get here, but you're finally here. At this point you need to add in the call to the TReK Application Programming Interface library. Remember, in this application you want a new value for MSID038 to show up in the main window once every second. To make this happen you need to call the `GetOneNewestConvertedIntegerValue` function in the TReK Application Programming Interface once every second to get the new value and then display it in the main window. In order to use the TReK API library you need to tell Visual C++ that you want this library to be linked into your application. In Visual C++, go to the **Project** menu and choose **Settings....** Look at the **Microsoft Foundation Classes** menu and change the selection so it is set to **Use MFC in a Shared DLL**. Your dialog should now look like the one shown in Figure 42.

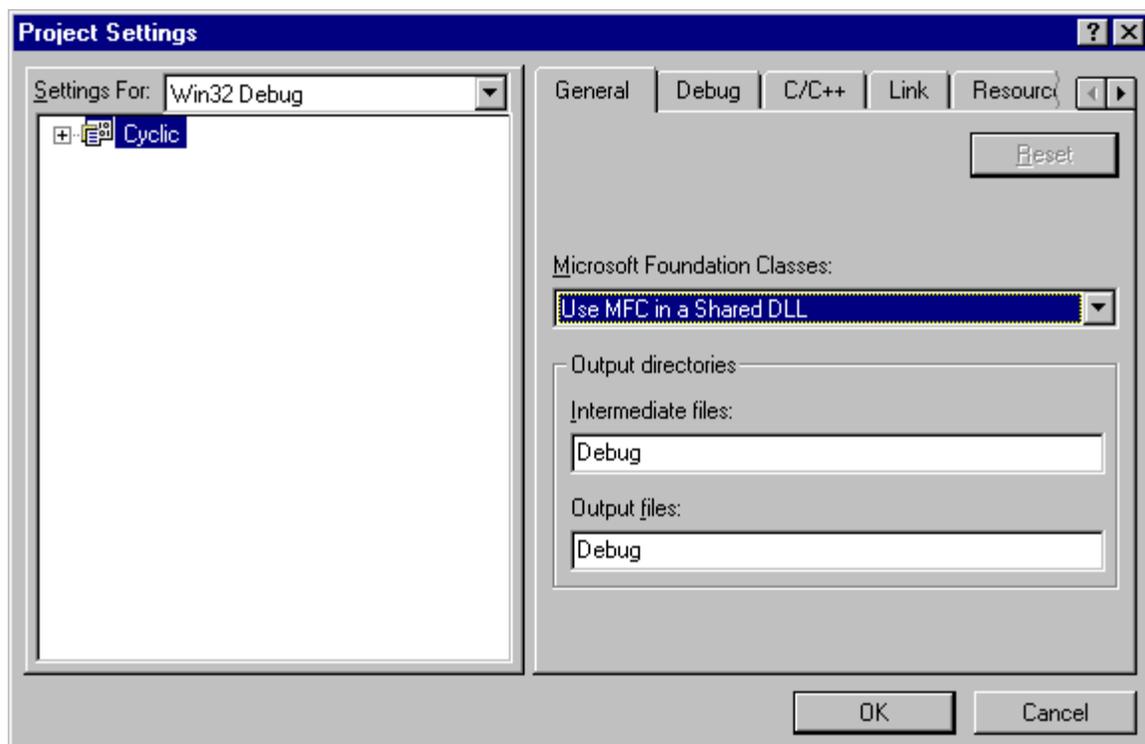


Figure 42 Project Settings Dialog After Modifications.

2. The next thing you need to do is to copy the `trek_user_api.lib` file into your project directory. This file is located in the TReK Installation directory under **lib**. Copy the `trek_user_api.lib` file into your Cyclic project directory. If you have worked with libraries before then you know that there is a corresponding `trek_user_api.dll` file. This file was installed in your `winnt\system32` directory when you installed the TReK software. Visual C++ knows how to find it so you don't need to do anything about the `trek_user_api.dll` file.
3. In the Project Settings dialog select the **Link** tab. In the **Object/library modules** field enter the location of the **`trek_user_api.lib`** file. Since you copied the `trek_user_api.lib` file into your project directory you should enter **`trek_user_api.lib`**. Figure 43 shows an example of what your dialog should look like now.

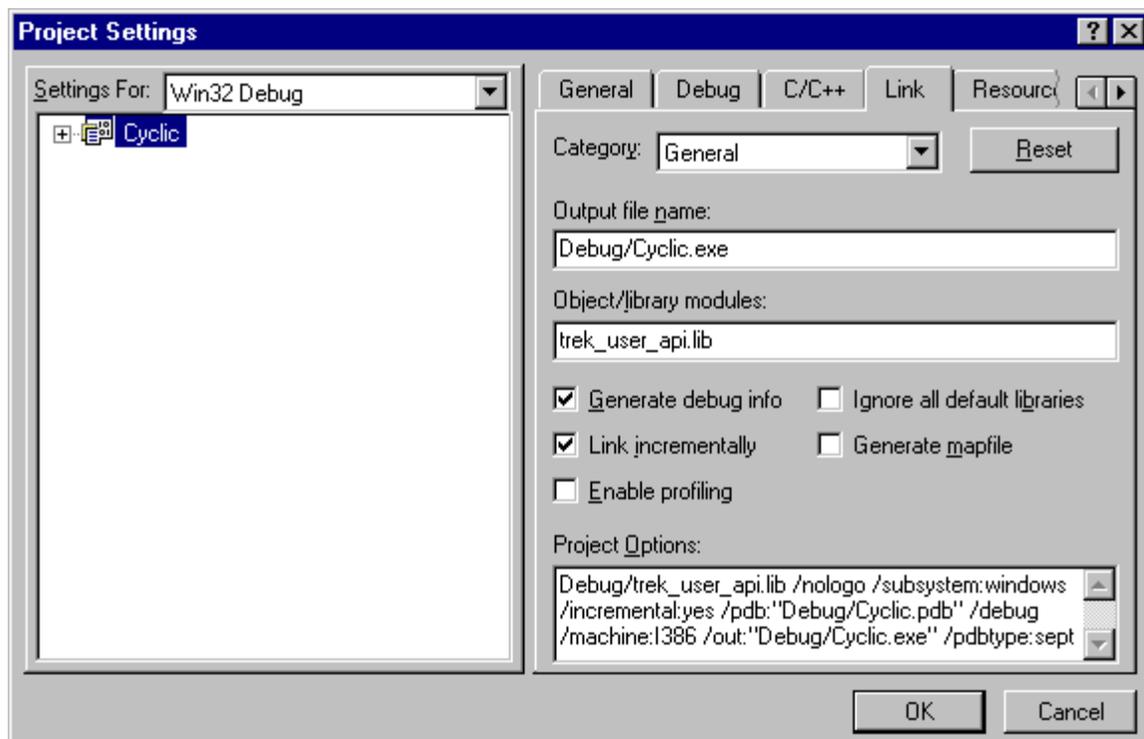


Figure 43 Project Settings Dialog showing reference to TReK API Library.

Note: If you want to run in Release mode, don't forget to do the following:

- In the Project Settings dialog under the **General** tab, make sure the **Settings For:** selection is set to **Win32 Release**. Look at the **Microsoft Foundation Classes** menu and make sure the selection is set to **Use MFC in a Shared DLL**.
  - In the Project Settings dialog on the **Link** tab change the **Settings For:** menu to **Win32 Release**. In the **Object/library modules** field enter the location of the `trek_user_api.lib` file which would be `trek_user_api.lib`.
4. Now is a good time to rebuild and execute the application just to make sure everything is okay. Once you have done this move on to the next step.
  5. You're almost done. You just have a few more steps to complete. Go to the FileView list in the Visual C++ window and double click on the `CyclicView.cpp` file. It's now time to fill in the **OnUpdate()**, **OnUpdateStart()**, and **OnUpdateStop()** message handlers.
  6. Locate the **OnUpdate()** function and make your function look like the following one. This function updates the main window with the newest MSID038 values.

```
void CCyclicView::OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint)
{
// TODO: Add your specialized code here and/or call the base
// class
    CCyclicDoc *pDoc = GetDocument();

    m_msid038_int_value = pDoc->msid038_int_value;
    m_msid038_int_status = pDoc->msid038_int_status;
    m_msid038_int_api = pDoc->msid038_int_api;

    UpdateData(FALSE);
}
```

7. Locate the **OnUpdateStart()** function and make your function look like the following one. This function calls the document class to create the timer thread. The timer thread will send a message to the View class once every second telling it to call the document class which will then call the TReK API to get the newest MSID038 data.

```
void CCyclicView::OnUpdateStart()
{
    // TODO: Add your command handler code here

    CCyclicDoc *pDoc = GetDocument();
    HWND view_handle;

    CUpdateInfo *info_ptr = new CUpdateInfo();

    view_handle = this->m_hWnd;

    info_ptr->view_handle = view_handle;

    // Set the update rate to 1000 milliseconds.
    info_ptr->update_rate = 1000;

    pDoc->StartUpdate(info_ptr);
}

```

8. Locate the **OnUpdateStop()** function and make your function look like the following one. This function kills the user interface thread so the display stops updating.

```
void CCyclicView::OnUpdateStop()
{
    // TODO: Add your command handler code here
    CCyclicDoc *pDoc = GetDocument();

    pDoc->StopUpdate();
}

```

9. Now you need to make the final updates to the Document class. In the Files list double click on the **CyclicDoc.cpp** file to open it. Find the list of include files located near the top of the file and add references to the following include files:

```
#include "trek.h"
#include "trek_error.h"
#include "trek_user_api.h"
```

Note: If these statements are added before the `#include "stdafx.h"` statement this will cause compile errors.

10. Now you need to make sure these files are located in your Cyclic project directory. You can copy these files out of the TREK installation directory. They are located under **include**. Copy the `trek.h`, `trek_error.h`, and `trek_user_api.h` files into your Cyclic project directory.

11. In the **CyclicDoc.cpp** file find the `CCyclicDoc::StartUpdate(CUpdateInfo *info_ptr)` function and update it so it matches the following function. When the user selects Start from the Update menu the StartUpdate message is sent. The StartUpdate message handler shown below creates the timer thread.

```
void CCyclicDoc::StartUpdate(CUpdateInfo *info_ptr)
{
    thread_ptr = new CUpdateThread(info_ptr);
    thread_ptr->CreateThread();

    thread_handle = thread_ptr->m_hThread;
}
```

12. In the **CyclicDoc.cpp** file find the `CCyclicDoc::StartUpdate()` function and update it so it matches the following function. When the user selects Stop from the Update menu the StopUpdate message is sent. The StopUpdate message handler shown below terminates the timer thread.

```
void CCyclicDoc::StopUpdate()
{
    DWORD dwExitCode = 0;

    TerminateThread( thread_handle, dwExitCode );

    delete thread_ptr;
}
```

13. In the **CyclicDoc.cpp** file find the `CCyclicDoc::GetNewData()` function and update it so it matches the following function. Remember you can copy this code out of the TReK examples directory. It is located in the CyclicDoc.cpp file in the examples\VisualC++\Cyclic directory.

```
void CCyclicDoc::GetNewData()
{
    // Declare Variables used with API Functions.
    int stream;
    int data_mode;
    char identifier[MSID_LEN];
    char apid[20];
    int return_code;
    long integer_value;
    char status[STATUS_LEN];
    int limit_es_flag;

    // Set up generic input variables.
    stream = PDSS_PAYLOAD;
    data_mode = REAL_TIME;
    limit_es_flag = LES_SENSE;
    strcpy(apid, "");
    strcpy(identifier, "MSID038");
    strcpy(msid038_int_api, "");

    // Call the GetNewestConvertedIntegerValue function in
    // the User API.
    return_code = GetOneNewestConvertedIntegerValue(stream,
                                                    identifier,
                                                    apid,
                                                    data_mode,
                                                    limit_es_flag,
                                                    msid038_con_token,
```

```
        &integer_value,  
        status);  
  
    if (return_code == SUCCESS)  
    {  
        // Copy the data into the document class member variables.  
        msid038_int_value = integer_value;  
        msid038_int_status = status;  
    }  
  
    GetAPIReturnCodeAsString(return_code, 70, msid038_int_api);  
}
```

14. That's it. This concludes Part IV. All you need to do now is to rebuild and execute the application. Don't forget to start the Telemetry Processing application before you run your Cyclic Display. MSID038 is in Packet ID 7. Packet ID 7 is defined in the TReK Telemetry Database (TelemetryDatabase.mdb). Be sure to Activate this packet in the Telemetry Processing application and send it from the Training Simulator application. If you don't you won't see any data.

## 5 Some Final Notes About This Tutorial

- Please remember that it is not a good idea to force the termination of a thread using `TerminateThread`. This does not give the thread a chance to clean up any resources it may be using. Please see your Microsoft documentation for information on how to send a message to the thread to tell it to exit.
- During this tutorial we had you copy several files out of the TReK Installation directory into your own Cyclic project directory. It is probably a better idea to reference these files instead of copying them. That way, when you receive a new set of TReK files, your projects will always be pointing to the latest version. Otherwise when you get new TReK files you will need to copy the new version of these files into each of your project directories. However, either way will work. To tell Visual C++ how to find libraries and header files you can use the Visual C++ Options dialog. If you go to the **Tools** menu and choose **Options...** the dialog in Figure 44 will appear. This dialog provides a way to tell Visual C++ where to look for include files and library files. Make sure the **Show Directories for:** menu has **Include files** selected. Select the empty frame and type in the entire path for the TReK **include** directory. Library files can be handled in the same way. Just change the **Show Directories for:** menu to Library files and type in the path for the TReK **lib** directory.

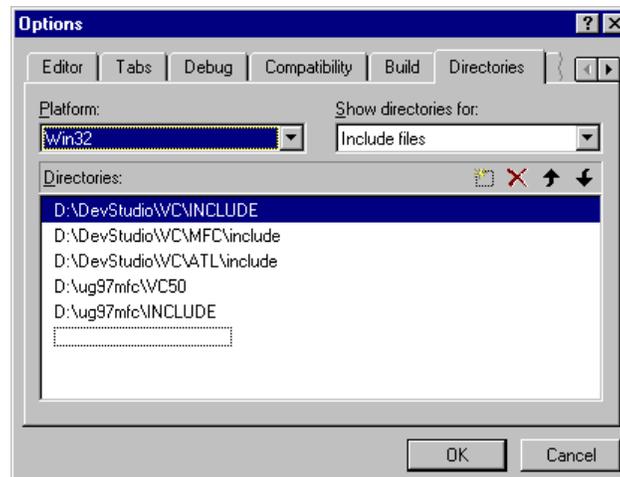


Figure 44 Options Dialog.

- The program you created during this tutorial can be modified and used to create other displays. The timer thread files can easily be used in other Visual C++ applications that need to update telemetry data in a cyclic fashion. When you get ready to build a

new display, you can save some time by starting with a copy of the Cyclic display and making the necessary modifications. In general, you would make modifications in the following areas:

1. Change the controls in the main window (and their corresponding member variables) to display a different set of parameters (MSIDs).
2. Update the View class so it matches the controls and member variables added in step 1 above.
3. Update the Document class so the correct API calls are made for the parameters added in step1 above.

Well.... That's All Folks!!!! This concludes the How to Build a Visual C++ Display tutorial. We hope you learned a lot and had a good time!

## Appendix A Glossary

Note: This Glossary is global to all TReK documentation. All entries listed may not be referenced within this document.

Application Programming Interface (API)	A set of functions used by an application program to provide access to a system's capabilities.
Application Process Identifier (APID)	An 11-bit field in the CCSDS primary packet header that identifies the source-destination pair for ISS packets. The type bit in the primary header tells you whether the APID is a payload or system source-destination.
Calibration	The transformation of a parameter to a desired physical unit or text state code.
Communications Outage Recorder	System that captures and stores payload science, health and status, and ancillary data during TDRSS zone of exclusion.
Consultative Committee for Space Data Systems (CCSDS) format	Data formatted in accordance with recommendations or standards of the CCSDS.
Consultative Committee for Space Data Systems (CCSDS) packet	A source packet comprised of a 6-octet CCSDS defined primary header followed by an optional secondary header and source data, which together may not exceed 65535 octets.
Conversion	Transformation of downlinked spacecraft data types to ground system platform data types.
Custom Data Packet	A packet containing a subset of parameters that can be selected by the user at the time of request.
Cyclic Display Update Mode	A continuous update of parameters for a particular display.
Decommutation (Decom)	Extraction of a parameter from telemetry.
Discrete Values	Telemetry values that have states (e.g., on or off).

Dump	During periods when communications with the spacecraft are unavailable, data is recorded onboard and played back during the next period when communications resume. This data, as it is being recorded onboard, is encoded with an onboard embedded time and is referred to as dump data.
Enhanced HOSC System (EHS)	Upgraded support capabilities of the HOSC systems to provide multi-functional support for multiple projects. It incorporates all systems required to perform data acquisition and distribution, telemetry processing, command services, database services, mission support services, and system monitor and control services.
Exception Monitoring	A background process capable of continuously monitoring selected parameters for Limit or Expected State violations. Violation notification is provided through a text message.
Expected State Sensing	Process of detecting a text state code generator in an off-nominal state.
EXPRESS	An EXPRESS Rack is a standardized payload rack system that transports, stores and supports experiments aboard the International Space Station. EXPRESS stands for EXpedite the PRocessing of Experiments to the Space Station.
File transfer protocol (ftp)	Protocol to deliver file-structured information from one host to another.
Flight ancillary data	A set of selected core system data and payload health and status data collected by the USOS Payload MDM, used by experimenters to interpret payload experiment results.

Grayed out	Refers to a menu item that has been made insensitive, which is visually shown by making the menu text gray rather than black. Items that are grayed out are not currently available.
Greenwich Mean Time (GMT)	The solar time for the meridian passing through Greenwich, England. It is used as a basis for calculating time throughout most of the world.
Ground ancillary data	A set of selected core system data and payload health and status data collected by the POIC, which is used by experimenters to interpret payload experiment results. Ground Ancillary Data can also contain computed parameters (pseudos).
Ground receipt time	Time of packet origination. The time from the IRIG-B time signal received.
Ground Support Equipment (GSE)	GSE refers to equipment that is brought in by the user (i.e. equipment that is not provided by the POIC).
Ground Support Equipment Packet	A CCSDS Packet that contains data extracted from any of the data processed by the Supporting Facility and the format of the packet is defined in the Supporting Facility's telemetry database.
Huntsville Operations Support Center (HOSC)	A facility located at the Marshall Space Flight Center (MSFC) that provides scientists and engineers the tools necessary for monitoring, commanding, and controlling various elements of space vehicle, payload, and science experiments. Support consists of real-time operations planning and analysis, inter- and intra-center ground operations coordination, facility and data system resource planning and scheduling, data systems monitor and control operations, and data flow coordination.

IMAQ ASCII	A packet type that was added to TReK to support a very specific application related to NASA's Return to Flight activities. It is not applicable to ISS. It is used to interface with an infrared camera that communicates via ASCII data.
Limit Sensing	Process of detecting caution and warning conditions for a parameter with a numerical value.
Line Outage Recorder Playback	A capability provided by White Sands Complex (WSC) to play back tapes generated at WSC during ground system communication outages.
Measurement Stimulus Identifier (MSID)	Equivalent to a parameter.
Monitoring	A parameter value is checked for sensing violations. A message is generated if the value is out of limits or out of an expected state.
Parameter	TReK uses the generic term parameter to mean any piece of data within a packet. Sometimes called a measurement or MSID in POIC terminology.
Payload Data Library (PDL)	An application that provides the interface for the user to specify which capabilities and requirements are needed to command and control his payload.
Payload Data Services Systems (PDSS)	The data distribution system for ISS. Able to route data based upon user to any of a number of destinations.
Payload Health and Status Data	Information originating at a payload that reveals the payload's operational condition, resource usage, and its safety/anomaly conditions that could result in damage to the payload, its environment or the crew.
Payload Operations Integration Center (POIC)	Manages the execution of on-orbit ISS payloads and payload support systems in coordination/unison with distributed International Partner Payload Control Centers, Telescience Support Centers (TSC's) and payload-unique remote facilities.

Payload Rack Checkout Unit (PRCU)	The Payload Rack Checkout Unit is used to verify payload to International Space Station interfaces for U.S. Payloads.
Playback	Data retrieved from some recording medium and transmitted to one or more users.
Pseudo Telemetry (pseudo data)	Values that are created from calculations instead of directly transported telemetry data. This pseudo data can be created from computations or scripts and can be displayed on the local PC.
Remotely Generated Command	A command sent by a remote user whose content is in a raw bit pattern format. The commands differ from predefined or modifiable commands in that the content is not stored in the POIC Project Command Database (PCDB).
Science data	Sensor or computational data generated by payloads for the purpose of conducting scientific experiments.
Subset	A collection of parameters from the total parameter set that is bounded as an integer number of octets but does not constitute the packet itself. A mini-packet.
Super sampled	A parameter is super sampled if it occurs more than once in a packet.
Swap Type	A flag in the Parameter Table of the TReK database that indicates if the specified datatype is byte swapped (B), word swapped (W), byte and word swapped (X), byte reversal (R), word reversal (V) or has no swapping (N).
Switching	A parameter's value can be used to switch between different calibration and sensing sets. There are two types of switching on TReK: range and state code.

Transmission Control Protocol (TCP)	TCP is a connection-oriented protocol that guarantees delivery of data.
Transmission Control Protocol (TCP) Client	A TCP Client initiates the TCP connection to connect to the other party.
Transmission Control Protocol (TCP) Server	A TCP Server waits for (and accepts connections from) the other party.
Telemetry	Transmission of data collected from a source in space to a ground support facility. Telemetry is downlink only.
Telescience Support Center (TSC)	A TSC is a NASA funded facility that provides the capability to plan and operate on-orbit facility class payloads and experiments, other payloads and experiments, and instruments.
User Application	Any end-user developed software program that uses the TREK Application Programming Interface software. Used synonymously with User Product.
User Data Summary Message (UDSM)	Packet type sent by PDSS that contains information on the number of packets sent during a given time frame for a PDSS Payload packet. For details on UDSM packets, see the POIC to Generic User IDD (SSP-50305).
Uplink format	The bit pattern of the command or file uplinked.
User Datagram Protocol (UDP)	UDP is a connection-less oriented protocol that does not guarantee delivery of data. In the TCP/IP protocol suite, the UDP provides the primary mechanism that application programs use to send datagrams to other application programs. In addition to the data sent, each UDP message contains both a destination port number and a fully qualified source and destination addresses making it possible for the UDP software on the destination to deliver the message to the correct recipient process and for the recipient process to send a reply.

User Product	Any end-user developed software program that uses the TReK Application Programming Interface software. Used synonymously with User Application.
Web	Term used to indicate access via HTTP protocol; also referred to as the World Wide Web (WWW).

## Appendix B Acronyms

Note: This acronym list is global to all TReK documentation. Some acronyms listed may not be referenced within this document.

AOS	Acquisition of Signal
API	Application Programming Interface
APID	Application Process Identifier
ASCII	American Standard Code for Information Interchange
CAR	Command Acceptance Response
CAR1	First Command Acceptance Response
CAR2	Second Command Acceptance Response
CCSDS	Consultative Committee for Space Data Systems
CDB	Command Database
CDP	Custom Data Packet
COR	Communication Outage Recorder
COTS	Commercial-off-the-shelf
CRR	Command Reaction Response
DSM	Data Storage Manager
EHS	Enhanced Huntsville Operations Support Center (HOSC)
ERIS	EHS Remote Interface System
ERR	EHS Receipt Response
EXPRESS	Expediting the Process of Experiments to the Space Station
ES	Expected State
FAQ	Frequently Asked Question
FDP	Functionally Distributed Processor
FSV	Flight System Verifier
FSV1	First Flight System Verifier
FSV2	Second Flight System Verifier
FPD	Flight Projects Directorate
FTP	File Transfer Protocol
GMT	Greenwich Mean Time
GRT	Ground Receipt Time
GSE	Ground Support Equipment
HOSC	Huntsville Operations Support Center
ICD	Interface Control Document
IMAQ ASCII	Image Acquisition ASCII
IP	Internet Protocol
ISS	International Space Station
LDP	Logical Data Path
LES	Limit/Expected State
LOR	Line Outage Recorder
LOS	Loss of Signal
MCC-H	Mission Control Center – Houston
MOP	Mission, Operational Support Mode, and Project
MSFC	Marshall Space Flight Center

MSID	Measurement Stimulus Identifier
NASA	National Aeronautics and Space Administration
OCDB	Operational Command Database
OS	Operating System
PC	Personal Computer, also Polynomial Coefficient
PCDB	POIC Project Command Database
PDL	Payload Data Library
PDSS	Payload Data Services System
PGUIDD	POIC to Generic User Interface Definition Document
POIC	Payload Operations Integration Center
PP	Point Pair
PRCU	Payload Rack Checkout Unit
PSIV	Payload Software Integration and Verification
RPSM	Retrieval Processing Summary Message
SC	State Code
SCS	Suitcase Simulator
SSP	Space Station Program
SSCC	Space Station Control Center
SSPF	Space Station Processing Facility
TCP	Transmission Control Protocol
TReK	Telescience Resource Kit
TRR	TReK Receipt Response
TSC	Telescience Support Center
UDP	User Datagram Protocol
UDSM	User Data Summary Message
URL	Uniform Resource Locator
USOS	United States On-Orbit Segment
VCDU	Virtual Channel Data Unit
VCR	Video Cassette Recorder
VPN	Virtual Private Network