

# Real-Time Graphics Tools for Windows

## Revision 3.0

Windows NT  
Windows 95  
Windows 98

### Revision 3.0 includes 32-bit DLLs only

**32-bit DLLs** - for Visual C++ 5.0 and higher Borland Delphi 2.0 and higher, and Visual Basic 5.0 and higher.

**No Runtime Royalties** - when used to create application programs.

### Presentation-Quality Hardcopy Output

**Scrolling Graph Types** - scrolling line, filled line, scatter plot, bar, logic, and sweep graphs. Graphs scroll horizontally or vertically, in any direction. Fixed and variable time increments can be used for update. Up to 32 traces per graph can be displayed. Scrolling lines, filled lines, bars and scatter plot symbols can be combined in the same graph.

**XY Plots** - real-time plotting of one analog value against another. Channels can be represented using lines and up to 10 different scatter plot symbols.

**Meters** - needle, arc, and pie.

**Text Displays** - annunciators, messages, and dynamic text. Annunciators can have unique alarm limits for each channel.

**Control Objects\*** - customized scroll bars, buttons, and LED indicators.

**Axes Labels** - display and update time axis labels in real-time in numeric or time formats.

### PID Control

**Alarm Checking and Reporting\*** - data sets can have global alarms, and individual channel alarms.

**Historical Data Replay** - scan through previously acquired data.

### Built-In Dialog Boxes for Editing Chart Characteristics

**Export Charts to Other Windows Applications** - graphs can be exported as metafiles and bitmaps

**Built-in Mouse Support\*** - click on chart objects like dynamic plots, axes, titles, legends, and automatically load the associated dialog box.

**C++, Microsoft Foundation Class (MFC)** - example programs are provided. The **Graphics Class Libraries (GCL)** for MFC, available as an add-on product, integrate these tools with the MFC Document/View architecture. See the **GCL** data sheets for detailed information.

**Documentation and Example Programs** - the software includes a comprehensive user manual with many programming examples, and more than 20 complete demo programs (on disk).

**Source Code Available** - see WIN-BMC-202, WIN-BD-202 and WIN-VB-202 product descriptions.



Four real-time applications running simultaneously.

The Quinn-Curtis, Inc. **Real-Time Graphics Tools for Windows** are a programming toolkit that harnesses the power of Windows providing you with the most complete set of real-time display functions available. This product includes 32-bit DLLs that can be used to create 32-bit applications for Windows NT, Windows 95 and Windows 98.

The software package contains functions for displaying scrolling graphs, sweep graphs, logic graphs, annunciators, xy plots, dynamic text, dynamic bar graphs, and meters. Plus you can add LED controls, buttons, scroll bars, and check boxes to any graphics window. Once the charts are created, they can be sent to Windows supported output devices at the resolution of the device, creating presentation quality printouts, slides, or transparencies.

The **Charting Tools for Windows** are included with the **Real-Time Graphics Tools**. This combination provides the most complete set of plotting functions available to Windows programmers. There are more than 500 functions, with over 600 pages of documentation and numerous example programs. Refer to the **Charting Tools for Windows** data sheets for a complete description of all static charting functions available.

### ORDERING INFORMATION

PART #	DESCRIPTION	PRICE
WIN-BMC-200	Real-Time Graphics Tools for C	\$600
WIN-BMC-201	Real-Time Graphics DLL Source C	\$600
WIN-BMC-202	Tools and Source for C	\$1200
WIN-BD-200	Real-Time Graphics Tools for Delphi	\$600
WIN-BD-202	Tools and Source for Delphi	\$1200
WIN-VB-200	Real-Time Graphics Tools for VB	\$500
WIN-VB-202	Tools and Source for VB	\$1100

### SHIPPING CHARGES

UPS Ground	UPS Blue	UPS Red	DHL	Canada
15	24	36	56	20

## Product Descriptions

WIN-BMC-200

DLL

### **Real-Time Graphics Tools for C/C++**

This product contains both the **Charting Tools for Windows DLL** and the **Real-Time Graphics Tools 32-bit DLLs**, a user manual for the **Charting Tools for Windows**, a user manual for the **Real-Time Graphics Tools**, C interface files, and example programs for Visual C/C++. The source code for the **Real-Time Graphics Tools DLL** is sold separately as the software product WIN-BMC-201.

WIN-BMC-201

Source

### **Real-Time Graphics Tools for C/C++**

This product contains the commented source code for the Quinn-Curtis **Real-Time Graphics Tools DLL** and the **Charting Tools for Windows DLL**. It cannot be used unless the user also has support files which are part of the WIN-BMC-200 software package.

WIN-BMC-202

DLL + Source

### **Real-Time Graphics Tools for C/C++**

Combines the WIN-BMC-200 and the WIN-BMC-201 software.

WIN-VB-200

DLL

### **Real-Time Graphics Tools for Visual Basic**

Contains user manuals, 32-bit DLLs, and demo programs for Visual Basic.

WIN-VB-202

DLL + Source

### **Real-Time Graphics Tools for Visual Basic**

Source code for the Visual Basic DLL is written in C.

WIN-BD-200

DLL

### **Real-Time Graphics Tools for Delphi**

Contains user manuals, 32-bit DLLs, and demo programs for Borland Delphi.

WIN-BD-202

DLL + Source

### **Real-Time Graphics Tools for Delphi**

Source code for the Borland Delphi DLL is written in C.

## System Requirements

The **Real-Time Graphics Tools for Windows** run on the IBM AT family of computers. Compiling a program using these tools requires:

- A computer configured for one of the following operating systems: Windows NT, Windows 95 or Windows 98.
- A complete installation of either the Visual C++, Delphi or Visual Basic compiler.
- 32-bit application programs will run on systems capable of running Windows NT, Windows 95 or Windows 98.

## Introduction to Real-Time Graphics Tools

### Terminology

#### Handle

You will see the term handle referenced throughout these data sheets as well as Windows documentation. Handles are simply numbers used to identify objects of any kind. A handle can be interpreted as a unique name given to an object. After a handle is associated with an object, you can always reference that object by its handle.

Microsoft Windows uses handles to reference windows, menus, dialog boxes, memory blocks, GDI objects, etc. In addition to that, in the Quinn-Curtis **Real-Time Graphics Tools** handles are used to reference data sets, graphical objects and pictures consisting of many graphical objects.

#### Page

Page is an entity based on a window containing one or more graphs as child windows. If page windows are created with the recommended default style, as pop up overlapped windows, they are movable and sizable. They can be printed or copied to the Clipboard. They may have a title bar and a sizable window frame. They can be minimized and maximized. A page window may have a menu bar. A page window can be owned by an application's main window or any other user created window. Up to sixty pages can simultaneously exist in one application.

#### Graph

Graph is an entity based on a child window belonging to a parent page window. It may contain a single plotting area and static and dynamic graphical objects, like axes, alarm messages, plotted data, text, etc. Every page can contain up to sixteen graphs. A graph window can be individually printed or copied to the Clipboard. Graphs belonging to the same page can be placed next to each other or superimposed.



Page consisting of four graphs. Dynamic and static graphical objects are mixed freely in the same graph.

## Plotting Area

A plotting area is a rectangle placed somewhere in the graph window. It is usually associated with at least one pair of axes. The axes extents are bounded by a plotting area. All data plots are drawn inside a plotting area and are clipped by the plotting area rectangle.

## Graphical Objects

Every graph consists of a number of graphical elements, or objects. Every graphical object is identified by the graph to which it belongs, and a handle. There are static and dynamic graphical objects. Static graphical objects are those that can be created by the *Charting Tools for Windows*. Dynamic graphical objects, like scrolling lines, dynamic bars, meters, etc., can be created with the *Real-Time Graphics Tools*. They display changing data in real time. Static and dynamic objects can be freely mixed in the same graph. There is no practical limit on the number of objects in a graph.

## Coordinate Systems

Dimensions and positions of windows and graphical objects can be expressed in different units based on the chosen coordinate system. Three different coordinate systems are supported - Page Normalized, Graph Normalized, and Physical. They all are device independent.

## Dynamic Data Sets

A dynamic data set is an object containing values of changing variables, alarm limits, channel tag names, and some other information. Real-time graphics functions, like **WRSetScrollGraph**, **WRSetMeter**, etc., work with dynamic data sets. Every dynamic graphical object, like a scrolling graph or dynamic bar graph, is created, initialized, and connected to a real-time data set with a corresponding setup function. Dynamic data sets are updated when the new data samples are acquired. Once the dynamic data set is provided with the new values, all the dynamic graphical objects connected to this data set are automatically updated.

## Adding Real-Time Graphs to Your Program

### Creating a Page Window

Graphics pages are created with the function **WGCreatePage**. It can be issued anywhere in the application program's main window procedure, or in any other user window procedure. Parameters passed to **WGCreatePage** specify the page window characteristics.

Three different types of image scaling can be chosen when a page is created - fixed, proportionate, and isotropic. If the fixed mode is chosen, the sizes of the graphs and their graphical objects will always remain the same regardless of the page window size. For example, if the window is reduced in size, the image is clipped. In proportionate mode, the sizes of all the graphs and their graphical objects will be always proportionate to the page window size. The aspect ratio will not be preserved. In isotropic mode, the sizes of all the graphs and their graphical objects will change when the page window is resized, but the original aspect ratio is maintained. This mode is the most appropriate one for sizing graph windows containing round graphical objects, such as meters or LED indicators.

## Creating a Graph Window

In order to draw a graph in the graph window the user must write a graph building procedure containing a sequence of calls to graphical object drawing functions. The graph building procedure is called from the graph window procedure that is invisible to the user. The syntax of this user written procedure must be as in the following example:

```
void FAR DrawPlG1 (PGRAPH_DEF pGrDesc, HDC hdc);  
  
Sub DrawPlG1 (pGrDesc as Long, hdc as Integer)
```

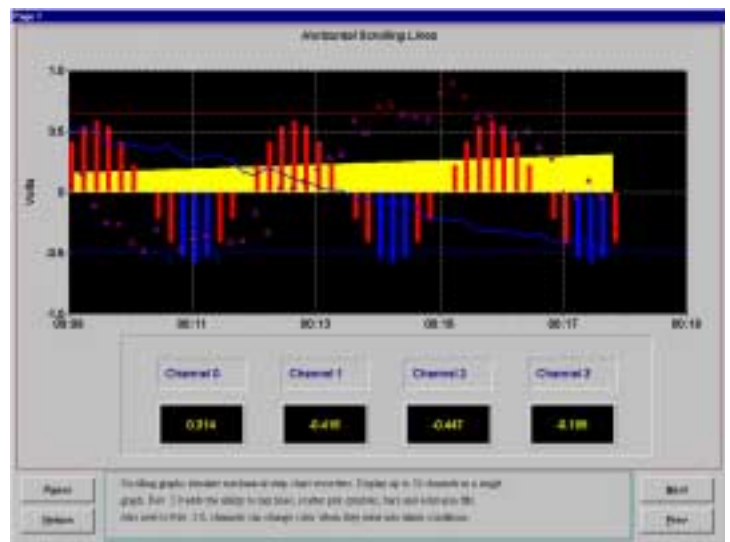
The first Quinn-Curtis function called from the graph building procedure should normally be **WGSetPlotArea**. (It is not required only if the chart has no axes based plots, for example, it only contains a meter.) It determines the size, position, and the background color of the plotting area. If axes are a part of the graph, a call to **WGScalePlotArea** is necessary. Then follows a sequence of calls to drawing functions building the graph. No actual drawing takes place at this time. What really happens - the graphical objects are created, and all the information passed to these functions is saved in the graphical objects' descriptors. After the building function finishes its job, the graph window is forced to update, and that is the time when actual drawing takes place.

## Elements of a Graph

### Axes

Like in the *Charting Tools for Windows*, graphs can be drawn by scaling the plotting area to the exact ranges (**WGScalePlotArea**), choosing the X- and Y-intercepts (**WGSetXYIntercepts**), drawing the X- and Y-axes with the specified number of tick marks (**WGDrawXAxis**, **WGDrawYAxis**) and labeling the major tick marks (**WGLabelAxis**).

In real-time plots one of the axes usually represents time. If you want to label the time axis in time format, then it should be created with the function **WRDrawTimeAxis** and labeled with the function **WRLabelTimeAxis**.



Combination scroll graphs can combine lines, filled lines, bars and scatter plot symbols. Channels can change color if alarm limits are exceeded.

## Labels

Time axis created with the function **WRDrawTimeAxis** can be labeled as numbers or in one of the time formats (HH:MM:SS, HH:MM, MM:SS). Time labels are created with the function **WRLabelTimeAxis**. They can be updated automatically or by request.

## Grids and Alarm Lines

Alarm and grid lines can be partially erased by moving objects, such as scrolling lines or dynamic bars. This can be detrimental for a graph appearance. To prevent that, grid and alarm lines objects are placed on the list maintained by an internal update timer routine and can be redrawn at user specified time intervals.

## Plotting Data

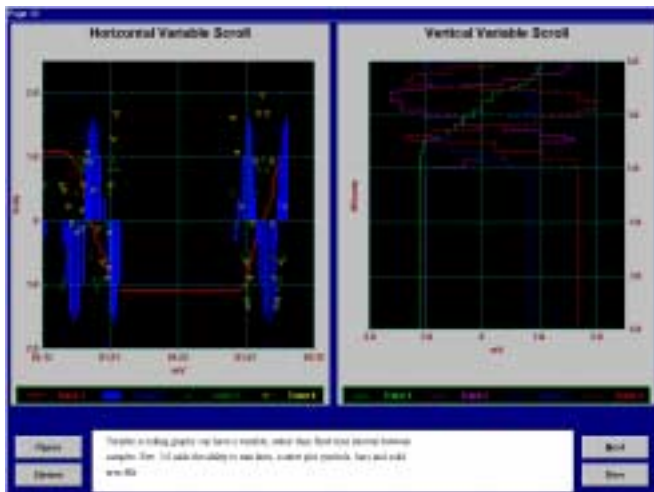
### Basic Dynamic Data Plot Types

There are nine basic dynamic data plot types: scrolling lines, scrolling bars, logic graphs, sweep graphs, dynamic bar graphs, dynamic text, text messages, alarm messages, and meters.

Some of the basic dynamic data plotting functions have additional modes that create variations on the basic plot types.

**Scrolling and Sweep Graphs** - Scrolling and sweep graphs are used to display analog data in the same fashion as an oscilloscope or strip chart recorder. These are basic dynamic plot types in which adjacent data points in a plot are connected with straight lines or with staircase steps. One graph displays data of all the channels in the selected dynamic data set, resulting in up to 32 traces. Scrolling graphs can be updated using a fixed or variable time increment. The orientation of the graph is either horizontal or vertical. The line color, style, and thickness are programmable.

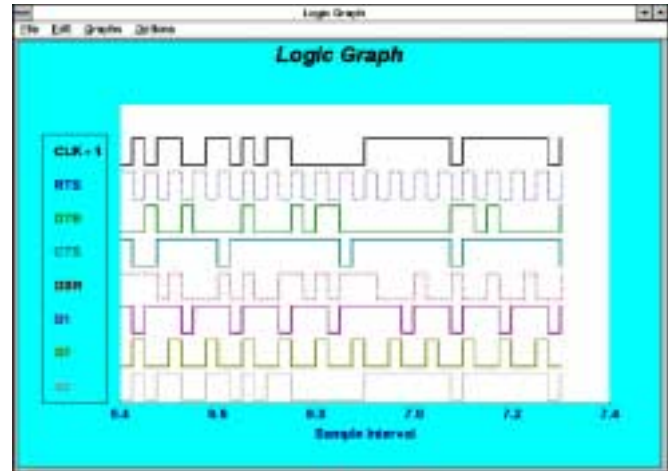
A scrolling graph draws dynamic traces in the direction of increasing time. Once the traces have reached the edge of the plotting area, they are reset (moved in the direction of the past time) by a specified amount. For example, if the relative reset is 50%, the scrolling graph will be shifted back by half of the plotting area width each time it fills the plotting area.



*Variable scrolling graphs use a variable time interval between samples. Variable scroll graphs have line, filled line, bar and scatter symbol options.*

Traces are not shifted in a sweep graph. Instead, this kind of graph displays a sweep line perpendicular to the time axis that moves across the graph in the direction of the increasing time. The latest data values are displayed immediately behind the line, while the points in front of the line represent the oldest values. The points in front of the line are erased as the sweep line passes over them.

**Logic Graphs** - A logic graph is modeled after the display of a logic analyzer. The primary use of the logic graph is to display binary data across the screen as a digital waveform. The orientation of the graph is either horizontal or vertical. The line color, style, and thickness are programmable.



*Logic graphs are ideal for displaying binary data associated with communications protocol analyzers.*

**Scrolling Bar Graph** - A scrolling bar graph displays data as a time series of moving bars. Scrolling bar graphs can display one or more channels of a dynamic data set. The scrolling bar graph function displays bars with one end of each bar starting at the same center point value.

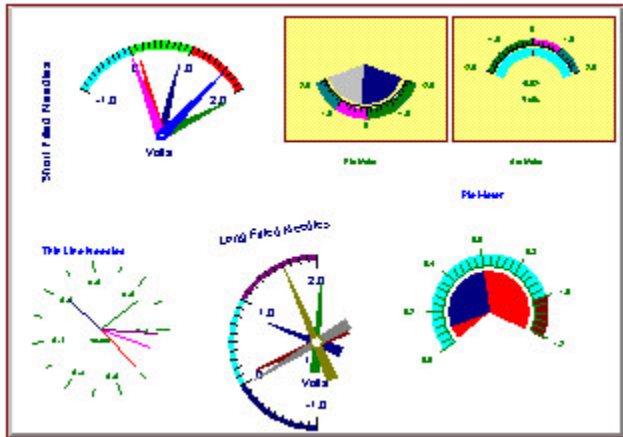
**XY Plots** - A real-time XY plot displays pairs of changing analog values in a fashion similar to driving the horizontal and vertical inputs of an oscilloscope with different signals. It is used for plotting one signal against another. One plot displays data of all the channels in the selected dynamic data set, resulting in up to 16 traces. The line color, style and thickness are programmable.

### Combination Scroll, Variable Scroll, Sweep, XY Graphs

- A combination graph will display the channels of a dataset as lines, bars, filled lines, and scatter plot symbols. It is possible to mix and match these types within the same graph. There are four main combination graph types: scrolling, variable scrolling, sweep and xy plot.

**Dynamic Bar Graphs** - In dynamic bar graphs every channel of a data set is represented by a bar changing its size. One end of each bar is always anchored at the specified center point value. Bars can change their size either in vertical or horizontal direction. They are drawn using "dithered" RGB colors and therefore thousands of colors are available on regular VGA systems. Standard Windows hatch styles can be used in bar graphs.

**Meters** - Meters display analog data in analog form. Familiar examples of meters are voltmeter or car speedometer. Three meter types are supported: arc, pie, and needle meters. Normal, High, High-High, Low and Low-Low alarm arcs can be drawn on a meter. A digital readout of the one can also be displayed.



Customize meters for any application. (metafile)

### Compound Dynamic Data Plot Types

There are five compound dynamic data plot types: auto scrolling graphs, auto bar graphs, PID bar graphs, annunciator panels, and automatic text displays. Only one compound dynamic plot can exist in a graph window.

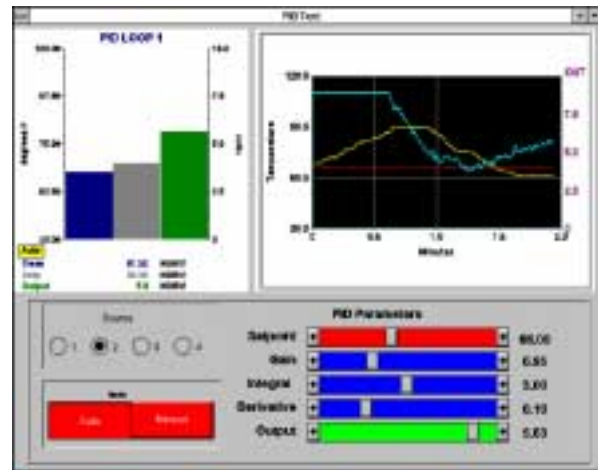
**Auto Scrolling Graphs** - The automatic scrolling graph combines a scrolling lines, scrolling bars, or sweep graph with formatted axes, titles, alarm lines, channel tag names, and current values.

**Auto Bar Graphs** - The automatic bar graph combines a dynamic bar graph with formatted axes, titles, alarm marks, alarm messages, channel tag names, and current values.

The **PID Bar Graph** is designed specifically for the display of PID loop parameters. It contains three vertical bars representing the current values of the process variable, set point, and PID output. It also has formatted vertical axes, titles, alarm marks, alarm messages, and current values of the process variable, setpoint, and PID output.

**Annunciator Panels** - Annunciator panels are used to display the current values and alarm states of real-time data in the text form. They consist of one or more rows of rectangular cells representing individual channels of a dynamic data set. Each cell can contain the tag name, units, current value, and alarm status message. If a channel is in alarm, the background color of the corresponding cell changes to a user specified high or low alarm color. Alarms limits can be set for annunciators on a channel by channel basis.

**Automatic Text Displays** - Automatic text displays present information about all the channels in a dynamic data set as a text table. Every row of the table corresponds to one channel. Columns correspond to channel tag names, current values, engineering units, setpoint, alarm messages, and descriptors.



One of our sample programs is an example of a PID tuning.

### Fonts and Text Attributes

Fonts used for all kinds of text in graphs are selected using **WGSetTextParams** and **WGSetTextByName** functions. The former uses font family for font selection. Font family describes the appearance of a font in a general way and can be used to specify font when it is not known if a certain typeface is available. The **WGSetTextByName** function specifies the exact typeface of the font.

**Bad Data Handling** - Revision 3.0 now checks the data passed to **WRUpdateData** for validity. First the data is checked to see if it represents a valid IEEE floating point number. If it passes this test, it is compared to the range specified by the **WRSetDynDataValidRange** function. This range is global to the entire application. If a data value is invalid, it will not be displayed in the real-time graphs. In line plots bad values cause a gap in the line. In the case of bars and scatter plot symbols the drawing of the associated bar or symbol is skipped. In the case of real-time text the string, **\*\*\*.\*** is displayed with the appropriate precision. Data checking only takes place on data passed to **WRUpdateData**.

Chan 1	Chan 2	Chan 3	Chan 4	Chan 5	Chan 6	Chan 7
1000	1000	642	378	870	600	870
vols	vols	vols	vols	vols	vols	vols
OK	OK	OK	OK	LOW	OK	OK
Chan 8	Chan 9	Chan 10	Chan 11	Chan 12	Chan 13	Chan 14
1000	1000	800	321	870	600	870
vols	vols	vols	vols	vols	vols	vols
OK	OK	OK	OK	OK	LOW	OK
Chan 15	Chan 16	Chan 17	Chan 18	Chan 19	Chan 20	Chan 21
8	1000	1000	320	870	600	870
vols	vols	vols	vols	vols	vols	vols
LOW	OK	OK	LOW	LOW	LOW	LOW
Chan 22	Chan 23	Chan 24	Chan 25	Chan 26	Chan 27	Chan 28
1000	1000	1000	1000	1000	1000	1000
vols	vols	vols	vols	vols	vols	vols
OK	OK	OK	OK	OK	LOW	OK

Notify operators of alarm conditions with annunciators.

## Special Features

### DDE

These tools support Dynamic Data Exchange. An application built with these tools can be a DDE client and display data provided by DDE Server applications, such as I/O drivers.

### Exchange of Images

Graphic images created with these tools can be exported to other applications via the Clipboard (functions **WGCOPYPage** and **WGCOPYGraph**), Windows Device Independent Bitmaps (**WGSavePageDIB**, **WGSaveGraphDIB**) or Windows Metafiles (**WGSavePageMeta**). Images copied to the Clipboard can be pasted to documents in other applications. Images saved as metafiles can be imported into an application directly.

### Printing Graphs

A page with all its graphs, or a single selected graph, can be printed on any printer or other hardcopy device supported by Windows. The driver for the printer must be installed and selected from the Windows Control Panel prior to an attempt to print.

### Error Handling

Various types of errors can occur when *Real-Time Graphics Tools* functions are called. For example, one or more arguments passed to a function may have illegal values, or system resources may not be available. Most of the functions in this package give an indication if they were successful. An error code is assigned to every error. It can be obtained by calling function **WGGetLastError**.

### Mouse Interaction

If any mouse related event occurs while a mouse cursor is over a graph window, the control is transferred to the function **WGGraphMouseEvent**. The function **WGGraphMouseEvent** is found in the file RTHOOK.C. By default this function processes the left mouse button click and double click events. In the first case, if the mouse cursor is positioned over a graphical object at the time of the click, the object is visually highlighted. In case of the double click, the default dialog box for editing the object appears. Mouse movement can be tracked with functions **WGGetMousePos** and **WGGetMousePosNorm**. They return current mouse cursor coordinates.

### Editing Graphs

The *Real-Time Graphics Tools* provide a number of dialog boxes for editing dynamic graphical objects. By default, double clicking the left mouse button, when the mouse cursor is over an object, will bring up an appropriate dialog box. There are also dialog boxes for editing parameters of dynamic data sets and PID control loops. All the dialog boxes provided in the *Charting Tools for Windows* are available.

### Window Management

Windows have to be repainted when they have been uncovered or resized. *Real-Time Graphics Tools* take care of repainting page and graph windows in a manner completely transparent to the user. Only once, in chart building procedures described above, the user defines the content of the

graph window in terms of graphical objects such as axes, plots, text, etc. All the graph-related window message processing is done transparently for the user.

The page window function, after performing necessary operations in response to certain messages, gives control to the function **WGUserPageProc** that can be found in the RTHOOK.C file. The user is free to add message and command processing to this function.

### Cleaning up

The section of an application program's main window procedure that processes the WM\_DESTROY message must contain a call to the **WRCleanup** function that frees all internally allocated memory and resources. Of course, it is the user's responsibility to free all the memory he allocated directly.

## Miscellaneous Compiler Support Features

### Floating Point Number Data Types

Type *realtyp*e is used for all floating point numbers. By default, it is defined to be equivalent to *double* but can be changed to *float*.

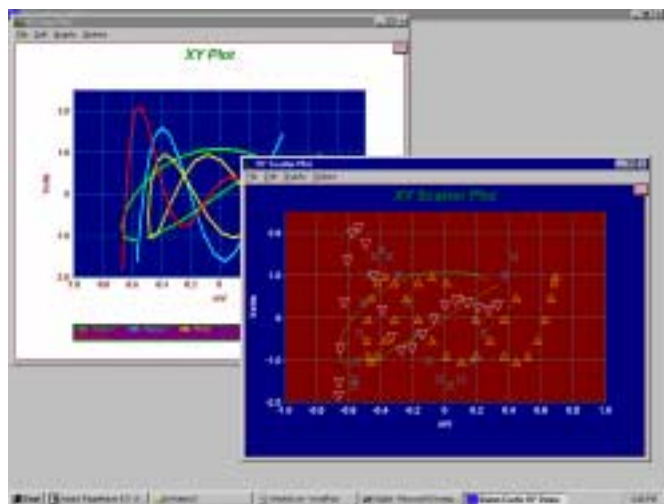
### C++ Support

The Real-Time Graphics Tools have been written using standard ANSI C syntax. But they are fully compatible with C++ and can be used for developing applications under C++. They are also compatible with the Microsoft Foundation Class. Example programs are provided.

*Edit chart characteristics with built-in dialog boxes.*

## Plotting Functions

Plotting functions can be envisioned as the graphical building blocks which are assembled on-screen to produce the complete graph. These functions create and initialize dynamic graphical objects. Refer to the *Windows Charting Tools* data sheets for a complete description of all static functions available.



*Use lines or scatter plot symbols in real-time XY plots.*

## Axes, Legends, and Labels

**WRDrawTimeAxis** - Draws a horizontal or vertical time axis with tick marks for the plotting area of the specified graph.

**WRDynGrid** - Draws dynamic grid lines perpendicular to the specified axis over the specified graph's plotting area.

**WRDynLegend** - Draws string legends for the specified dynamic graphical object. Each legend identifies a single channel with a text string and, optionally, a graphical symbol representing the plot type.

**WRDynLegendPtr** - Same as **WRDynLegend**, but uses a pointer instead of a memory handle.

**WRLabelTimeAxis** - Writes a sequence of labels in time or numeric format at the tick marks of the specified time axis.

**WRLogicTags** - Displays text tags for all channels of the specified logic graph.

**WRLogicTagsPtr** - Same as **WRLogicTags**, but uses a pointer instead of a memory handle.

## Displaying Data

**WRSetAlarmLines** - Draws a set of up to five lines corresponding to the values of setpoint, high, low, high high and low low alarms for the specified dynamic data set in the specified graph's window.

**WRSetAlarmMarks** - Draws a set of up to five labeled marks corresponding to the values of setpoint, high, low, high high and low low alarms for the specified dynamic data set. The marks are drawn next to the specified axis, usually in combination with bar graphs.

**WRSetAlarmText** - Displays alarm strings for all channels or one channel of the specified dynamic data set.

**WRSetAnnun** - Creates an Annunciator Panel display for the specified dynamic data set.

**WRSetAnnun3** - This function is almost identical to the **WRSetAnnun** function with the difference being that the initial font size is passed in as an argument (as in **WRSetAnnun2**), and second, that low and high alarm colors are passed in, rather than just a single alarm color used for both high and low.

**WRSetAutoBarGraph** - Creates a dynamic bar graph modeled after process control style faceplates. Dynamic values of the specified data set are represented by bars and as dynamic text at the side of the plotting area.

**WRSetAutoScrollGraph** - Creates and formats a scrolling lines, scrolling bars, or sweep graph combined with axes, titles, alarm lines, channel tag names, and current values.

**WRSetAutoText** - Automatically creates formatted dynamic text display.

**WRSetBarGraph** - Creates a dynamic bar graph for the specified data set.

**WRSetDynBitBlitMode** - This function sets the dynamic bit block transfer (BitBlt) mode for the specified graph window. If the BitBlt mode for a scrolling graph window is set to TRUE, whenever the graph resets, a new image is drawn in an offscreen bit map, then copied to the display screen. This minimizes flicker. If the BitBlt mode for a scrolling graph is FALSE, whenever a graph resets the old lines are erased on the display screen by writing over them in the background color of the plotting area, then new lines are drawn in place of them.

**WRSetDynLineGraph** - This function creates a dynamic line graph for the specified data set.

**WRSetDynText** - Displays dynamic numerical values for the specified dynamic data set.

**WRSetLogicGraph** - Creates a logic graph for the specified data set.

**WRSetMessage** - Creates up to eight different text messages for the specified dynamic data set. Only one of the messages is displayed at any time. The messages are triggered by binary values of the data set.

**WRSetArcMeter** - Creates an arc meter.

**WRSetMeter** - Creates a needle meter.

**WRSetMeterBorder** - Modifies the appearance of the meter previously created with the function **WGSetMeter**,

**WGSetArcMeter**, or **WGPieMeter**.

**WRSetPieMeter** - Creates a pie meter.

**WRSetPIDBarGraph** - Creates a PID bar graph for the specified PID object.

**WRSetScrollBarGraph** - Creates a Scrolling Bars plot for the specified dynamic data set.

**WRSetScrollChannelAlarmColors** - The color of the lines, bars, and scatter plot symbols in Scrolling Graphs, Variable Scrolling Graphs, Sweep Graphs can be made to change in response to global alarm conditions for a dataset. In scrolling lines, only the portion of the line in alarm changes color. For scrolling scatter plots and scrolling bar graphs the bar or symbol which is in alarm changes color.

**WRSetScrollComboGraph** - This function creates a real-time scrolling combo graph in the specified graph's window. A combo graph is a scrolling graph where each channel of a dataset can be displayed as a unique line, bar, filled line or scatter plot symbol.

**WRSetScrollMultiBarGraph** - This function creates a real-time scrolling multi-bar graph in the specified graph's window. A multi-bar graph is a scrolling graph where each channel of a dataset can be displayed as a unique bar. It is created by setting up a combo scrolling graph with MK\_BAR selected for each channel.

**WRSetScrollGraph** - Creates a real-time scrolling graph for the specified dynamic data set.

**WRSetScrollScatterGraph** - This function creates a real-time scrolling scatter graph in the specified graph's window. A real-time scatter graph is a scrolling graph where each channel of a dataset can be displayed as a unique scatter plot symbol. It is created by setting up a scrolling combo graph with a scatter plot symbol selected for each channel.

**WRSetSweepGraph** - Creates a sweeping line plot for the specified dynamic data set.

**WRSetSweepComboGraph** - This function creates a real-time combo sweep graph in the specified graph's window. A combo sweep graph is a sweep graph where each channel of a dataset can be displayed as a unique line, bar, filled line or scatter plot symbol.

**WRSetSweepScatterGraph** - This function creates a real-time sweep scatter graph in the specified graph's window. A real-time sweep scatter graph is a sweep graph where each channel of a dataset can be displayed as a unique scatter plot symbol. It is created by setting up a sweep combo graph with a scatter plot symbol selected for each channel.

**WRSetVarComboScroll** - This function creates a real-time combo variable scroll graph in the specified graph's window. A combo variable scroll graph is a variable scroll graph where each channel of a dataset can be displayed as a unique line, bar, filled line or scatter plot symbol. A variable scroll graph is a scrolling graph which uses a variable update time.

**WRSetXYGraph** - Creates a real-time XY graph.

**WRSetXYScatterGraph** - This function creates a real-time xy scatter plot in the specified graph's window. A xy scatter plot xy plot where each channel of a dataset can be displayed as a unique line or scatter plot symbol. NOTE: Bars and filled lines are not an allowable type in xy scatter plots.

## Custom Controls

**WRSetButtonControl** - Creates a button child control in the specified graph, and defines its logic of operation and appearance.

**WRSetButtonGroup** - Creates a group of mutually exclusive button child controls in the specified graph, and defines their appearance. Only one of the controls in a group can be selected.

**WRSetLEDControl** - Creates a button child control that looks like a Light Emitting Diode (LED) used in instrumentation.

**WRSetScrBarControl** - Creates a scroll bar child control in the specified graph.

## Alarm and Setpoint Functions

These functions manage alarm parameters and alarm checking.

**AlarmEventProc** - Application-defined alarm event callback function that processes alarm events sent to it when an alarm condition is detected.

**WREnableAlarmCheck** - Enables or disables alarm checking for the specified dynamic data set.

**WRSetAlarm** - Sets alarm limit and defines alarm message for the given alarm type for the specified dynamic data set.

**WRSetAlarmHyst** - Sets alarm hysteresis, or deadband, values.

**WRSetAlarmProc** - Registers the application-defined alarm callback function.

**WRSetSetpoint** - Sets the setpoint value for the specified dynamic data set.

## Process Control Functions

**WRCalcPID** - Calculates the PID algorithm output for the given PID loop.

**WRDefinePID** - Creates a PID loop (object) and specifies the PID algorithm parameters.

**WRDeletePID** - Destroys the specified PID loop and, optionally, the data sets associated with it.

## DDE Functions

These functions facilitate dynamic data exchange between applications.

**DdeProc** - Extension to the main DDE callback function.

**WRConnectDDE** - Registers an application with the Dynamic Data Exchange Management Library (DDEML), attempts to establish a DDE conversation with a server application, starts an advise loop for the specified data item and updates values in the dynamic data set.

**WRInitDde** - Registers an application with DDEML. It should be used only if an application uses direct calls to DDEML library in addition to Quinn-Curtis DDE functions.

**WRSetDdeProc** - Registers the application-defined DDE callback function.

## Dynamic Data Management Functions

Real-time data for plotting must be organized in objects called dynamic data sets. Data set management functions create and manipulate data sets.

**WRDefineDynDataSet** - Creates a dynamic data set representing changing "real-time" values for one or more data channels.

**WRFreeDynDataSet** - Destroys the dynamic data set.

**WRGetBinData** - Retrieves current binary data values for the specified dynamic data set.

**WRGetHistData** - Copies values of one channel of the specified dynamic data set from its historical buffer to a user array.

**WRGetLastData** - Retrieves latest and previous data values for all channels of the specified dynamic data set.

**WRGetMinMaxValues** - Determines the maximal and minimal values for one or all channels of the specified dynamic data set.

**WRSetDynDataValidRange** - Set the valid range for data. Values outside of the specified range is considered to be invalid.

## Real-Time Update Functions

These functions are used to update data sets and graphical objects in real time.

**WRAutoRescale** - Rescales a dynamic plot based on the actual range of previously collected data.

**WRMoveHistScroll** - Changes the starting value of the independent variable (time) for the specified scrolling graph which is in the history replay mode.

**WRReconnectDynDataSet** - Disconnects the specified graphical object from the current dynamic data set and connects it to another data set.

**WRResetHistBuffer** - Resets the historical buffer of a specified dynamic data set.

**WRResetScroll** - Resets a specified scrolling or XY graph so that the next update will start drawing from the initial position.

**WRResetTimeLabels** - Resets the time axis labels.

**WRResumeScroll** - Resumes real-time update of the scrolling graph object which is in the history replay mode.

**WRSetBit** - Sets or resets a specified digital channel bit of a dynamic data set.

**WRSetBitExclusive** - Sets the specified digital channel bit and resets all the others.

**WRShowHist** - Stops real-time scrolling and displays historical data from the buffer of the specified dynamic data set.

**WRStartTimeLabelsUpdate** - Initializes automatic update of time axis labels based on the reset of the associated scrolling graph.

**WRStopObjTimeUpdate** - Stops periodic update of the specified dynamic grids or alarm lines.

**WRStopTimeLabelsUpdate** - Stops automatic update of time axis labels based on the reset of the associated scrolling graph.

**WRUpdateTimeLabels** - Updates time axis labels by the specified time increment. It should be used when the labels are not updated automatically.

**WRUpdateBinaryData** - Updates the specified dynamic data set with new binary values and calls update functions for graphical objects using this data set.

**WRUpdateData** - Updates the specified dynamic data set with new values, saves the values in the historical buffer, and calls update functions for graphical objects using this data set.

**WRUpdateObj** - Immediately updates the specified dynamic graphical object.

### User Interface Support Functions

This group of functions allows the programmer to change and get the status of many parameters of graphical objects. They can be used to modify graphs after they have been created, and for building menus and dialog boxes.

#### Displaying Data

**WRChangeAlarmMarks** - Enables or disables display of the alarm or setpoint mark for the specified alarm type, changes the alarm mnemonic associated with the mark.

**WRChangeDynTextColor** - Changes the text color for the given channel of the specified dynamic text object.

**WRChangeScrollGraph** - Changes parameters of the specified scrolling graph object.

**WRChangeTimeLabelsFormat** - Changes format of time axis labels.

**WREnableAlarmLine** - Enables or disables display of the alarm or setpoint line for the specified alarm type.

**WRGetAlarmLineState** - Determines if the specified alarm line is enabled.

**WRGetAlarmMarks** - Determines if the specified alarm mark is enabled, returns the pointer to the alarm mnemonic.

**WRGetDynTextColor** - Retrieves the color code for the given channel of the specified dynamic text object.

**WRGetMessageParams** - Retrieves the text string and color for the specified message in a message object.

**WRGetScrollGraphParams** - Retrieves parameters of the specified scrolling graph object.

#### Meters

**WRChangeMeterAngles** - Changes the start and end points of the meter scale, in degrees.

**WRChangeMeterCenter** - Changes the relative position of the meter center.

**WRChangeMeterRange** - Changes the upper and lower limits and the center point value, in physical units, for the specified meter.

**WRChangeMeterTicks** - Changes the number, position, and length of the tick marks of the specified meter.

**WRGetMeterAngles** - Returns starting and ending angles.

**WRGetMeterCenter** - Returns meter center and diameter.

**WRGetMeterRange** - Returns the maximal, minimal, and center point values for the specified meter.

**WRGetMeterTicks** - Returns tick mark related parameters for the specified meter.

#### Custom Controls

**ButControlProc** - Application-defined callback function that is called when the button control changes its state.

**ScBarControlProc** - Application-defined callback function that is called when the position of the specified scroll bar control changes.

**WRChangeButtonColor** - Changes the color of the specified button control.

**WRChangeButtonState** - Changes the current state of the specified button control. It simulates pressing a button or checking a check box.

**WRChangeMessage** - Changes the text string and color for the specified message in a message object.

**WRChangeScrBar** - Changes parameters of the specified scroll bar child control.

**WRGetButtonColor** - Retrieves the current color of the specified button control.

**WRGetButtonState** - Returns the current state of the specified button control.

**WRGetScrBar** - Retrieves parameters of the specified scroll bar child control.

**WRGetScrollPos** - Retrieves the current position of the scroll box (thumb) of a scroll bar control, in physical units.

**WRSetScrollPos** - Sets the position of the scroll box (thumb) and redraws the specified scroll bar to reflect the new position of the scroll box.

#### Alarms

These functions manage alarm parameters and alarm checking. Every channel in a data set can have unique alarm limit settings.

**AlarmEventProc** - Application-defined alarm event callback function that processes alarm events sent to it when an alarm condition is detected.

**WRCopyChAlarmsToDataSetAlarms** - The alarm lines in a graph only display the global alarms for a dataset. If you need to see one particular set of channel alarms displayed in the current graph, call this function to copy a set of channels alarm values to the dataset global alarms.

**WREnableAlarmCheck** - Enables or disables alarm checking for the specified dynamic data set.

**WREnableAlarmCheckCh** - This function enables or disables alarm checking for the specified channel in a dynamic data set.

**WRSetAlarm** - Sets alarm limit and defines alarm message for the given alarm type for the specified dynamic data set.

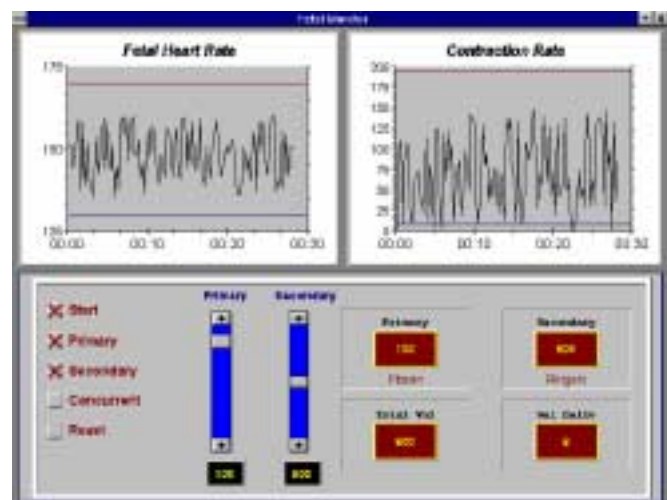
**WRSetAlarmCh** - This function sets the alarm limit value and message string for the specified channel, alarm type and dynamic dataset. The state of the alarm checking flag for the specified channel is set to TRUE after this call.

**WRSetAlarmHyst** - Sets alarm hysteresis, or deadband, values.

**WRSetAlarmHystCh** - This function sets alarm hysteresis, or dead band, values for the specified channel of a dynamic data set.

**WRSetAlarmProc** - Registers the application-defined alarm callback function.

**WRSetSetpoint** - Sets the setpoint value for the specified dynamic data set.



*The Real-Time Graphics Tools for Windows can greatly improve the user interface for many commercial instruments.*

## Child Controls

**WRChangeButtonColor** - Changes the color of the specified button control.

**WRChangeButtonState** - Changes the current state of the specified button control. It simulates pressing a button or checking a check box.

**WRChangeMessage** - Changes the text string and color for the specified message in a message object.

**WRGetButtonColor** - Retrieves the current color of the specified button control.

**WRGetButtonState** - Returns the current state of the specified button control.

## PID

**WRChangePIDAutoState** - Sets PID loop to manual or automatic mode.

**WRChangePIDConstants** - Changes the values of control constants of the specified PID loop.

**WRChangePIDOut** - Changes the output of PID loop.

**WRChangePIDSetp** - Changes the setpoint value of the specified PID loop.

**WRChangePIDSteadyState** - Changes the value of the steady state, or bias, of the specified PID loop.

**WREditPIDDlg** - Brings up the standard dialog box for editing the specified PID loop.

**WRGetPIDConstants** - Returns the values of the specified PID loop's control constants.

**WRGetPIDSetp** - Returns the value of the specified PID loop's setpoint.

**WRGetPIDAutoState** - Returns the auto/manual state of the specified PID loop.

## Annunciator Panels

**WRGetAnnAlm** - Retrieves the handle to the alarm text object for the specified channel of the annunciator.

**WRGetAnnTag** - Retrieves the handle to the tag name text object for the specified channel of the annunciator.

**WRGetAnnTitle** - Retrieves the handle to the title text object of the annunciator.

**WRGetAnnVal** - Retrieves the handle to the dynamic text object for the specified channel of the annunciator.

**WRGetAnnUnits** - Retrieves the handle to the engineering units text object for the specified channel of the annunciator.

## Miscellaneous Functions

**WRDeleteRTGraph** - Destroys the specified graph. All the graphical objects belonging to the graph are also destroyed and memory occupied by them is released. Dynamic grids and alarm lines are removed from the timer list. Data sets plotted in the graph are not affected.

**WRDeleteRTObject** - Deletes the specified dynamic graphical object and its descriptor, removes objects of types **GOD\_GRID** and **GOD\_ALARM\_LINES** from the timer list.

**WRGetDynTypeSize** - Returns the size of the descriptor of dynamic graphical objects of the specified type.

**WRCleanup** - Cleanup procedure that must be called by application before exit.

```
void FAR DrawPIG1 (PGRAPH_DEF pGrDesc, HDC hdc);
BOOL CALLBACK AboutProc (HWND hDlg, UINT message, WPARAM wParam,
LPARAM lParam);
void CALLBACK _export TimerRoutine (HWND hwnd, UINT msg, UINT
idTimer, DWORD dwTime);

/*****
FUNCTION: WinMain (HINSTANCE, HINSTANCE, LPSTR, int)
PURPOSE: calls initialization function, processes message loop
*****/
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
LPSTR lpCmdLine, int nCmdShow)
{
MSG msg;

if (!hPrevInstance)
if (!InitApplication(hInstance))
return (FALSE);
if (!InitInstance(hInstance, nCmdShow))
return (FALSE);
// Message loop
while (GetMessage(&msg, NULL, NULL, NULL))
{
TranslateMessage(&msg);
DispatchMessage(&msg);
};
return (msg.wParam);
}

/*****
FUNCTION: InitApplication (HINSTANCE)
PURPOSE: Initializes window data and registers window classes
*****/
BOOL InitApplication (HINSTANCE hInstance)
{
WNDCLASS wc;

// main window class
wc.style = CS_HREDRAW | CS_VREDRAW;
wc.lpszClassName = "MainWndProc";
wc.cbClsExtra = 0;
wc.cbWndExtra = 0;
wc.hInstance = hInstance;
wc.hIcon = LoadIcon(hInstance, "LOGO");
wc.hCursor = LoadCursor(NULL, IDC_ARROW);
wc.hbrBackground = (HBRUSH) GetStockObject(LTGRAY_BRUSH);
wc.lpszMenuName = NULL;
wc.lpszClassName = szAppName;

return (RegisterClass(&wc));
}

/*****
FUNCTION: InitInstance (HINSTANCE, int)
PURPOSE: Saves instance handle and creates main window
*****/
```

## C Programming Example

This program demonstrates how easy it is to add real-time graphics capability to a Windows program. Most of the work is done by the **DrawPIG1** function. A single, simple scroll graph with alarm lines is created inside a parent page window. The data is updated by the Windows timer.

```
//*****
//
// Description: C example building one graph with the
// Real-Time Graphics Tools for Windows
//
//*****
#include <windows.h>
#include <stdlib.h>
#include <math.h>

#include "qcwin.h"
#include "qcwrt.h"
#include "menus.h"

int idTimer;
HINSTANCE hInst;
HANDLE hScroll;
HANDLE hData;
PGRAPH_DEF pDynGrDesc = NULL;

char szAppName[] = "Demo";

// Prototypes of forward referenced functions
LRESULT CALLBACK MainWndProc (HWND hwnd, UINT message, WPARAM
wParam, LPARAM lParam);
BOOL InitApplication (HINSTANCE);
BOOL InitInstance (HINSTANCE, int);
void FAR StartGraphs1 (PPAGE_DEF pPageDesc);
```

```

*****
BOOL InitInstance (HINSTANCE hInstance, int nCmdShow)

```

```

{
    HWND hwnd;
    // save current instance - MUST BE HERE
    hInst = hInstance;

    hwnd = CreateWindow
    (
        szAppName, // Create main window
        "Strip Chart Emulation", // Main window title
        WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
        NULL,
        NULL,
        hInstance,
        NULL
    );

    if (!hwnd)
        return (FALSE);

    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);
    return (TRUE);
}

```

```

/*****
Main window procedure
*****/

```

```

LRESULT CALLBACK MainWndProc (HWND hwnd, UINT message,
    WPARAM wParam, LPARAM lParam)
{
    static PPAGE_DEF pPageDesc1 = NULL;
    HWND hwndPage;

    switch (message)
    {
        case WM_CREATE:
            pPageDesc1 = WGCreatePage("PAGE1", // page ID string
                hwnd, // handle to the parent window
                hInst, // application instance handle
                "Simulator", // Window title string
                StartGraphs1, // pointer to graph creation function
                NULL, // no menu
                C_LIGHTGRAY, // window background color
                MM_ISOTR, // window sizing mode
                OL, // window style
                PAGE_FULL,
                0, 0, 0, 0); // initial window size and position
                // if used (not used here)

            idTimer = SetTimer (NULL, 0, 250,
                (TIMERPROC)TimerRoutine); // start timer
            return 0;

        case WM_DESTROY:
            KillTimer (NULL, idTimer); // stop the timer
            WRCleanup(TRUE); // clean up real-time tools memory
            // and free data arrays
            PostQuitMessage(0);
            return 0;
    }

    return (DefWindowProc(hwnd, message, wParam, lParam));
}

```

```

/*****
Routine StartGraphs1 is called by the Quinn-Curtis
Windows Charting Tools when a page is created.
It must be filled by the user, normally with
functions WGCreateGraph that initialize individual graphs.
*****/

```

```

void FAR StartGraphs1(PPAGE_DEF pPageDesc)
{
    // Initialize graphs
    pDynGrDesc = WGCreateGraph (pPageDesc,
        DrawPlG1, // function which builds graph
        0.005, 0.005, // window relative position
        0.49, 0.8, //inside parent page window
        C_LIGHTGRAY, // light gray background
        C_WHITE, // white border
        1); // border width in pixels
}

```

```

/*****
Builds the graph using Q-C Real-Time Graphics Calls
*****/

```

```

void FAR DrawPlG1 (PGRAPH_DEF pGrDesc, HDC hdc)
{
    HANDLE hAxisX, hAxisY; // axes handles
    BOOL bFlags [NALMLINES];
    short nTraces = 3;
    short nLineColor [DYN_MAXVAL];
    short nLineWidth [DYN_MAXVAL];

```

```

short i, nGridUpdate = 4;
realtyp rSampleInt, rResetInt;
realtyp rHigh, rLow, rSetp;

// define the plotting area of the graph
WGSetPlotArea(pGrDesc, hdc, 0.24, 0.15, 0.95, 0.57, C_BLACK);

// scale the plotting area for an x range of 0 to 1.0
// and y range of -2.0 to 2.0
WGScalePlotArea (pGrDesc, 0.0, -2.0, 1.0, 2.0);

// set the intercepts to 0.0, -2.0
WGSetXYIntercepts (pGrDesc, 0.0, -2.0);

// axes to be drawn in solid, black, 1 pixels thick
WGSetLineStyle(pGrDesc, hdc, PS_SOLID, 1, C_CYAN);

// set current font to Arial, 8 points
WGSetTextByName (C_RED, "Arial", 8, 0);

// draw the x axis
hAxisX = WGDrawXAxis(pGrDesc, hdc, 0.2, 0, POS_MIDDLE);

// draw the y axis
hAxisY = WGDrawYAxis(pGrDesc, hdc, 1.0, 1, POS_LEFT);

// Label the x axis
WGLabelAxis(pGrDesc, hdc, hAxisX, POS_BELOW, NF_DECIMAL, 1,
    LL_ON, NULL);

WGSetTextByName (C_RED, "Arial", 8, TEXT_ITAL);

// Label the y axis
WGLabelAxis(pGrDesc, hdc, hAxisY, POS_LEFT, NF_DECIMAL, 1,
    LL_ON, NULL);

// Set the line style for the dynamic grids
WGSetLineStyle(pGrDesc, hdc, PS_DOT, 1, C_LIGHTCYAN);

// set current font to Arial, 8 points
WGSetTextByName (C_RED, "Arial", 8, TEXT_BOLD);

// Write axes titles
WGTitleAxis(pGrDesc, hdc, hAxisX, POS_BELOW, "Minutes");
WGTextNorm (pGrDesc, hdc, "Volts", 0.1, 0.35,
    TA_CENTER | TA_BOTTOM, TEXT_VERTLEFT);

// set current font
WGSetTextByName (C_GREEN, "Arial", 10, TEXT_BOLD | TEXT_ITAL);

// Write graph title
WGTitleGraph(pGrDesc, hdc, "Horizontal Strip Chart");

/*****
SET UP DYNAMIC PORTION OF GRAPH
Display a dynamic grid x and y axes
WRDynGrid(pGrDesc, hAxisX, GRID_MAJOR, nGridUpdate);
WRDynGrid(pGrDesc, hAxisY, GRID_MAJOR, nGridUpdate);

Assign each line style, width and color for the scroll graph
for (i = 0; i < nTraces; i++)
{
    nLineWidth [i] = 2;
}
nLineColor [0] = C_LIGHTRED;
nLineColor [1] = C_WHITE;
nLineColor [2] = C_LIGHTGREEN;

// Values for limits
rSampleInt = 0.02; rResetInt = 0.95;

// Define the dynamic data set
hData = WRDefineDynDataSet ("Scroll 1",
    nTraces, // # of traces
    "volts", // unit string
    NULL, // no tag name
    200); // history buffer = 200

hScroll = WRSetScrollGraph (pGrDesc,
    hData, // handle to dynamic data set
    rSampleInt, // sample interval
    rResetInt, // relative reset
    OR_HORZ, // lines are horizontal
    NO_STEP, // style - linear interp
    NULL,
    nLineWidth,
    nLineColor);

// set the flags for the alarm lines
bFlags[0] = TRUE;
bFlags[1] = TRUE;
bFlags[2] = TRUE;
bFlags[3] = FALSE;
bFlags[4] = FALSE;

```

## Visual Basic Programming Example

```
' *****
' ** Description: Visual Basic 16-bit example building one graph
' ** with the Real-Time Graphics Tools for Windows
' *****
' FEATURES OF THIS DEMO
'
' Scroll Graphs - horizontal, Dynamic Grids
' Dynamic Data sets, Timers, Alarms

Option Explicit

Const NT = 3
Dim hScroll As Integer ' handle to scrolling graph object
Dim hData As Integer ' handle to dynamic data set
Dim nCount As Integer ' update counter
Dim pPageDesc1 As Long ' Page descriptor
Dim pDynGrDesc As Long ' Graph descriptors

' *****
' ** Builds the graph using Q-C Real-Time Graphics Calls
' *****
Sub DrawPlG1 (pGrDesc As Long, hdc As Integer)
Dim hAxisX, hAxisY As Integer ' axis handles
Static bFlags(NALMLINES) As Integer
Dim nTraces As Integer, nErr As Integer
Static nLineColor(NT) As Integer
Static nLineWidth(NT) As Integer
Static nLineStyle(NT) As Integer
Dim i As Integer, nGridUpdate As Integer
Dim rSampleInt As Double, rResetInt As Double
Dim rHigh As Double, rLow As Double, rSetp As Double
Dim nText As Integer, hObj As Integer
Static nAlmLineStyle(NALMLINES) As Integer
Static nAlmLineWidth(NALMLINES) As Integer
Static nAlmLineColor(NALMLINES) As Integer

nTraces = 3
nGridUpdate = 4

Call WGSetsPlotArea(pGrDesc, hdc, .24, .15, .95, .57, C_BLACK)
' scale the plotting area for an x range of 0 to 1.0
' and y range of -3.0 to 3.0
Call WGSscalePlotArea(pGrDesc, 0#, -3#, 1#, 3#)
' set the intercepts to 0.0, -3.0
Call WGSsetXYIntercepts(pGrDesc, 0#, -3#)
' axes to be drawn in solid, black, 1 pixels thick
Call WGSsetLineStyle(pGrDesc, hdc, PS_SOLID, 1, C_CYAN)
' set current font to Arial, 7 points
nText = WGSsetTextByName(C_RED, "Arial", 8, 0)
' draw x axis - major ticks every 0.2 physical units, no minor ticks
hAxisX = WGDRAWXAxis(pGrDesc, hdc, .2, 0, POS_MIDDLE)
' draw y axis - major ticks every physical unit, and 1 minor tick
hAxisY = WGDRAWYAxis(pGrDesc, hdc, 1#, 1, POS_LEFT)
' Label the x axis, in dec. format, 1 digit after the decimal point.
hObj = WGLabelAxis(pGrDesc, hdc, hAxisX, POS_BELOW, NF_DECIMAL, 1,
LL_ON, "")
nText = WGSsetTextByName(C_RED, "Arial", 8, TEXT_ITAL)
' Label the y axis, in dec. format, 1 digit after the decimal point.
hObj = WGLabelAxis(pGrDesc, hdc, hAxisY, POS_LEFT, NF_DECIMAL, 1,
LL_ON, "")
' Set the line style for the dynamic grids
Call WGSsetLineStyle(pGrDesc, hdc, PS_DOT, 1, C_LIGHTCYAN)
' set current font to Arial, 8 points
nText = WGSsetTextByName(C_RED, "Arial", 8, TEXT_BOLD)
' Write axes titles
hObj = WGTITLEAxis(pGrDesc, hdc, hAxisX, POS_BELOW, "Minutes")
hObj = WGTITLEAxis(pGrDesc, hdc, hAxisY, POS_LEFT, "Volts", .1, .35, TA_CENTER Or
TA_BOTTOM, TEXT_VERTLEFT)
' set current font to Arial, 10 points, bold, italic
nText = WGSsetTextByName(C_GREEN, "Arial", 10, TEXT_BOLD Or TEXT_ITAL)
' Write graph title
hObj = WGTITLEGraph(pGrDesc, hdc, "Horizontal Strip Chart")

' *****
' SET UP DYNAMIC PORTION OF GRAPH
' Display a dynamic grid at the major tick interval for both
' x and y axes, updating the grid every 4 seconds.
hObj = WRDYNGrid(pGrDesc, hAxisX, GRID_MAJOR, nGridUpdate)
hObj = WRDYNGrid(pGrDesc, hAxisY, GRID_MAJOR, nGridUpdate)
' Assign each line width and color for the scroll graph
For i = 0 To nTraces - 1
nLineWidth(i) = 2
nLineStyle(i) = PS_SOLID
Next i
nLineColor(0) = C_LIGHTRED
nLineColor(1) = C_WHITE
nLineColor(2) = C_LIGHTGREEN

rSampleInt = .02
rResetInt = .95

' Define dynamic data set - units, no tags, history buffer size = 200
hData = WRDefineDynDataSet("Scroll 1", nTraces, "volts", "", 200)

' create scrolling lines - horizontal, style = linear interp
hScroll = WRSetScrollGraph(pGrDesc, hData, rSampleInt, rResetInt,
OR_HORZ, NO_STEP, nLineStyle(0), nLineWidth(0), nLineColor(0))

' set the flags for the alarm lines
bFlags(0) = True
bFlags(1) = True
bFlags(2) = True
bFlags(3) = False
bFlags(4) = False
For i = 0 To NALMLINES - 1
nAlmLineStyle(i) = PS_SOLID
nAlmLineWidth(i) = 2
Next i
' Assign colors for alarm lines
nAlmLineColor(0) = C_YELLOW
nAlmLineColor(1) = C_RED
nAlmLineColor(2) = C_BLUE
rHigh = .95 ' High
rLow = -1.4 ' Low
rSetp = .25 ' Setpoint

' Assign setpoint, low and high alarm values
nErr = WRSetSetpoint(hData, rSetp)
nErr = WRSetAlarm(hData, ALM_LOW, rLow, "LOW")
nErr = WRSetAlarm(hData, ALM_HIGH, rHigh, "HIGH")
' Draw alarm lines - alarm line updated every 2 seconds,
' lines are horizontal, line style = PS_SOLID
hObj = WRSetAlarmLines(pGrDesc, hData, 2, OR_HORZ, bFlags(0),
nAlmLineStyle(0), nAlmLineWidth(0), nAlmLineColor(0))
End Sub

' *****
' ** Page window is created when form is loaded
' *****
Sub Form_Load ()
Timer1.Interval = 250
nCount = 0

' page is created in current window as a child of the main window
pPageDesc1 = WGCreatPage("PAGE1", ScrollG1Form.hWnd, "Simulator",
"", C_LIGHTGRAY, MM_ISOTR, 0, PAGE_FULL, 0, 0, 0, 0, TopDesc)
If (pPageDesc1 <> 0) Then
Call StartGraphs1(pPageDesc1)
Call WGUUpdatePage(pPageDesc1)
End If
End Sub

Sub Form_Unload (Cancel As Integer)
Call WRcleanUp(TopDesc)
End Sub

' *****
' ** Calls WGCreatGraph to initialize graph
' *****
Sub StartGraphs1 (pPageDesc As Long)
Dim hdc As Integer

' first graph: light gray bg, white border, border width = 1
pDynGrDesc = WGCreatGraph(pPageDesc, .005, .005, .49, .8,
C_LIGHTGRAY, C_WHITE, 1, TopDesc, hdc)
' Call graph building procedure
Call DrawPlG1(pDynGrDesc, hdc)
End Sub

' *****
' ** Data is updated by a Timer Control
' *****
Sub Timer1_Timer ()
Dim rArg As Double
Static rNewVals(3) As Double

' Initialize the DLL every time the timer is called!
' Otherwise the DLL will not know who is calling it.
' Necessary only if multiple RT Tools applications or multiple
' instances of the same application can run simultaneously.
Call WGSStart(TopDesc)
If WGiDescValid(pDynGrDesc) Then
rArg = M_PI * nCount
' simulate data
rNewVals(0) = Sin(rArg / (1020# - nCount))
rNewVals(1) = Sin(rArg / 52#) * Rnd * 1.4
rNewVals(2) = Cos(rArg / 60#) * Rnd * .5
nCount = nCount + 1
If (nCount > 1000) Then
nCount = 0
End If
' Update the data set
Call WRUpdateData(hData, rNewVals(0), 0)
End If
End Sub
```