Randy Hyde's Win32 Assembly Language Tutorials (Featuring HOWL)

#3: Check Boxes

In this third tutorial of this series, we'll take a look at implementing check box buttons on HOWL forms. Specifically, we'll be looking at Windows' check box user-interface elements.

Prerequisites:

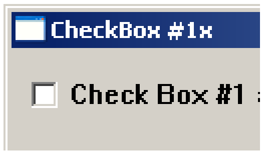
This tutorial set assumes that the reader is already familiar with assembly language programming and HLA programming in particular. If you are unfamiliar with assembly language programming or the High Level Assembler (HLA), you will want to grab a copy of my book "The Art of Assembly Language, 2nd Edition" from No Starch Press (www.nostarch.com). The HOWL (HLA Object Windows Library) also makes heavy use of HLA's object-oriented programming facilities. If you are unfamiliar with object-oriented programming in assembly language, you will want to check out the appropriate chapters in "The Art of Assembly Language" and in the HLA Reference Manual. Finally, HOWL is documented in the HLA Standard Library Reference Manual; you'll definitely want to have a copy of the chapter on HOWL available when working through this tutorial.

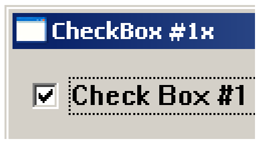
Source Code:

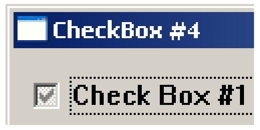
The source code for the examples appearing in this tutorial are available as part of the HLA Examples download. You'll find the sample code in the Win32/HOWL subdirectories in the unpacked examples download. This particular tutorial uses the files *005_checkbox1.hla, 006_checkbox2.hla, 007_checkbox3.hla.hla,* and *008_checkbox4.hla*. Though this particular document does not describe *005x_checkbox1.hla, 006x_checkbox2.hla, 007x_checkbox3.hla,* and *008x_checkbox4.hla,* , you may also find these files of interest when reading through this tutorial.

Check Boxes:

Checkboxes are a special type of button that have a binary (check/not checked) or trinary (checked/not checked/grayed) state associated with them. Generally, "clicking" on a check box will toggle the check box's state between the two or three different possible states.







The check box consists of two components on the form: the white rectangle is the actual check box (and will contain a check mark if in the true state, it will be empty if in the false state, and it will be a gray/checked box if in the third state). Note that only certain types of check boxes support the grayed state. In this first example, we're going to look at a binary state (checked/not checked) check box.

Because check boxes are special types of buttons, it should come as no surprise the that HOWL Declarative Language (HDL) syntax for a check box declaration is nearly identical to that of push buttons (that we looked at in the last chapter). Here is a typical HDL `wCheckbox` declaration:

```
wCheckBox
(
    checkBox1,                  // Field name in mainWindow object
    "Check Box #1 abcdefg",     // Caption for check box
    10,                         // x position
    10,                         // y position
    125,                        // width
    25,                         // height
    onClick1                    // "on click" event handler
)
```

Except for the name (`wCheckBox`), this declaration is identical to that of a `wPushButton` object:

```
wPushButton
(
    button2,                    // Field name in mainWindow object
    "Hide checkbox 1",          // Caption for push button
    175,                        // x position
    10,                         // y position
    150,                        // width
    25,                         // height
    hideShowCheckBox            // initial "on click" event handler
)
```

Although Windows considers them both buttons, check boxes are fundamentally different from push buttons. When the user clicks on a push button this is a signal to the application that some action should take place. Click boxes, on the other hand, typically use clicks to change their state and perform no other action. A typical onClick handler for a HOWL check box will be the following:

```
proc onClick1:widgetProc;
begin onClick1;

    // Invert the check in the check box:

    mov( thisPtr, esi );
    (type wCheckBox_t [esi]).get_check();
    xor( 1, eax );
    and( 1, eax );
    (type wCheckBox_t [esi]).set_check( eax );


end onClick1;
```

The `get_check` method call returns the current state of the check box (0/false = unchecked, 1/true = checked). The code above will invert the value read by `get_check` (the `xor` instruction does this) and then writes the new boolean value back to the check box via the `set_check` method call.

Note that the HOWL `wCheckBox` widget does not automatically process button clicks. That is, if you create a `wCheckBox` widget and don't provide an "on-click" handler for that widget, when the user clicks on the check box it will not automatically switch between states. It is possible to create automatic check box widgets, where Windows handles the checking and unchecking of the check box when the user clicks on it, but you'll have to use `wClickable` objects (see the discussion later in this tutorial) for that purpose.

Generally, when an application uses a check box, it calls `get_check` to obtain the state of the check box whenever it needs to test that value. This may occur long after the user has clicked on the check box to change the state (assuming they have clicked on it -- the user could have left the check box in its initial state).

The first demo program in this tutorial is a small modification of the `004_button3.hla` source file from the previous tutorial -- It simply swaps a checkbox object for the `button1` object on the form (plus a few textual changes to say "check box" rather than "button"). The only real modification is the `onClick1` widgetProc shown earlier. Here is the complete code for *005_checkbox1.hla*:

```
// CheckBox1-
//
//  This program demonstrates operations on checkboxes, including
// simulated checkbox clicks, double clicks, showing and hiding checkboxes,
// enabling and disabling checkboxes, moving checkboxes, and resizing checkboxes.

program checkBox1;
#linker( "comdlg32.lib" )
#linker( "comctl32.lib" )

?@NoDisplay     := true;
?@NoStackAlign  := true;

#includeOnce( "stdlib.hhf" )
#includeOnce( "howl.hhf" )

const
```

```
    applicationName := "CheckBox #1";
    formX           := w.CW_USEDEFAULT; // Let Windows position this guy
    formY           := w.CW_USEDEFAULT;
    formW           := 600;
    formH           := 600;



// Forward declarations for the onClick widgetProcs that we're going to
// call when a button is pressed.

proc onClick1               :widgetProc; @forward;
proc hideShowCheckBox       :widgetProc; @forward;
proc enableDisableCheckBox  :widgetProc; @forward;
proc moveCheckBox           :widgetProc; @forward;
proc resizeCheckBox         :widgetProc; @forward;
proc onDblClick             :widgetProc; @forward;
proc onQuit                 :widgetProc; @forward;



// Here's the main form definition for the app:

wForm( mainAppWindow );

    var
        showState   :boolean;
        b1Enabled   :boolean;
        align(4);

    wCheckBox
    (
        checkBox1,              // Field name in mainWindow object
        "Check Box #1 abcdefg", // Caption for check box
        10,                     // x position
        10,                     // y position
        125,                    // width
        25,                     // height
        onClick1                // "on click" event handler
    )


    wPushButton
    (
        button2,                // Field name in mainWindow object
        "Hide checkbox 1",      // Caption for push button
        175,                    // x position
        10,                     // y position
        150,                    // width
        25,                     // height
        hideShowCheckBox        // initial "on click" event handler
    )


    wPushButton
    (
        button3,                // Field name in mainWindow object
        "Disable checkbox 1",   // Caption for push button
        175,                    // x position
        40,                     // y position
        150,                    // width
        25,                     // height
        enableDisableCheckBox   // initial "on click" event handler
    )


    wPushButton
    (
        button4,                // Field name in mainWindow object
        "Move checkbox 1",      // Caption for push button
        175,                    // x position
        70,                     // y position
        150,                    // width
        25,                     // height
        moveCheckBox            // initial "on click" event handler
    )


    wPushButton
    (
        button5,                // Field name in mainWindow object
        "Resize checkbox 1",    // Caption for push button
        175,                    // x position
        100,                    // y position
        150,                    // width
        25,                     // height
        resizeCheckBox          // initial "on click" event handler
    )


    wPushButton
    (
        button6,                // Field name in mainWindow object
        "DblClick to Click",    // Caption for push button
        175,                    // x position
        130,                    // y position
        150,                    // width
        25,                     // height
        NULL                    // no single click handler
    )


    // Place a quit button in the lower-right-hand corner of the form:

    wPushButton
    (
        quitButton,             // Field name in mainWindow object
        "Quit",                 // Caption for push button
        450,                    // x position
        525,                    // y position
        125,                    // width
        25,                     // height
        onQuit                  // "on click" event handler
    )

endwForm
```

```
// Must invoke the following macro to emit the code generated by
// the wForm macro:

mainAppWindow_implementation();








// The onDblClick widget proc will handle a double click on button6
// and simulate a single click on checkBox1.

proc onDblClick:widgetProc;
begin onDblClick;

    mov( mainAppWindow.checkBox1, esi );
    (type wCheckBox_t [esi]).click();

end onDblClick;



// The resizeCheckBox widget proc will resize checkBox1 between widths 125 and 150.

proc resizeCheckBox:widgetProc;
begin resizeCheckBox;

    mov( mainAppWindow.checkBox1, esi );
    (type wCheckBox_t [esi]).get_width();
    if( eax = 125 ) then

        stdout.put( "Resizing check box to width 150" nl );
        (type wCheckBox_t [esi]).set_width( 150 );

    else

        stdout.put( "Resizing check box to width 125" nl );
        (type wCheckBox_t [esi]).set_width( 125 );

    endif;

end resizeCheckBox;



// The moveCheckBox widget proc will move checkBox1 between y positions 10 and 40.

proc moveCheckBox:widgetProc;
begin moveCheckBox;

    mov( mainAppWindow.checkBox1, esi );
    (type wCheckBox_t [esi]).get_y();
    if( eax = 10 ) then

        stdout.put( "Moving check box to y-position 40" nl );
        (type wCheckBox_t [esi]).set_y( 40 );
    else

        stdout.put( "Moving check box to y-position 10" nl );
        (type wCheckBox_t [esi]).set_y( 10 );

    endif;

end moveCheckBox;



// The enableDisableCheckBox widget proc will hide and show checkBox1.

proc enableDisableCheckBox:widgetProc;
begin enableDisableCheckBox;

    mov( thisPtr, esi );
    if( mainAppWindow.b1Enabled ) then

        (type wCheckBox_t [esi]).set_text( "Enable checkbox 1" );
        mov( false, mainAppWindow.b1Enabled );
        stdout.put( "Disabling button 1" nl );
        mov( mainAppWindow.checkBox1, esi );
        (type wCheckBox_t [esi]).disable();

    else

        (type wCheckBox_t [esi]).set_text( "Disable checkbox 1" );
        mov( true, mainAppWindow.b1Enabled );
        stdout.put( "Enabling button 1" nl );
        mov( mainAppWindow.checkBox1, esi );
        (type wCheckBox_t [esi]).enable();

    endif;

end enableDisableCheckBox;



// The hideShowCheckBox widget proc will hide and show checkBox1.

proc hideShowCheckBox:widgetProc;
begin hideShowCheckBox;

    mov( thisPtr, esi );
    if( mainAppWindow.showState ) then

        (type wCheckBox_t [esi]).set_text( "Hide checkbox 1" );
        mov( false, mainAppWindow.showState );
        stdout.put( "Showing button 1" nl );
        mov( mainAppWindow.checkBox1, esi );
        (type wCheckBox_t [esi]).show();

    else

        (type wCheckBox_t [esi]).set_text( "Show checkbox 1" );
        mov( true, mainAppWindow.showState );
        stdout.put( "Hiding button 1" nl );
        mov( mainAppWindow.checkBox1, esi );
        (type wCheckBox_t [esi]).hide();

    endif;

end hideShowCheckBox;




// Here's the onClick handler for the checkbox. As it's a simple
// 2-state checkbox, simply invert the state every time it's clicked.


proc onClick1:widgetProc;
begin onClick1;

    // Invert the check in the check box:

    mov( thisPtr, esi );
    (type wCheckBox_t [esi]).get_check();
    xor( 1, eax );
    and( 1, eax );
    (type wCheckBox_t [esi]).set_check( eax );


end onClick1;



// Here's the onClick event handler for our quit button on the form.
// This handler will simply quit the application:

proc onQuit:widgetProc;
begin onQuit;

    // Quit the app:

    w.PostQuitMessage( 0 );

end onQuit;






// We'll use the main application form's onCreate method to initialize
// the various buttons on the form.
//
// This could be done in appStart, but better to leave appStart mainly
// as boilerplate code. Also, putting this code here allows us to use
// "this" to access the mainAppWindow fields (a minor convenience).

method mainAppWindow_t.onCreate;
var
    thisPtr :dword;

begin onCreate;

    mov( esi, thisPtr );
    // Initialize the showState and enableDisableButton data fields:

    mov( false, this.showState );
    mov( true, this.b1Enabled );

    // Set up button6's onDblClick handler:

    mov( thisPtr, esi );
    mov( this.button6, esi );
    (type wPushButton_t [esi]).set_onDblClick( &onDblClick );

end onCreate;



///////////////////////////////////////////////////////////////////////////////
//
//
// The following is mostly boilerplate code for all apps (about the only thing
// you would change is the size of the main app's form)
//
//
///////////////////////////////////////////////////////////////////////////////
//
// When the main application window closes, we need to terminate the
// application. This overridden method handles that situation.  Notice the
// override declaration for onClose in the wForm declaration given earlier.
// Without that, mainAppWindow_t would default to using the wVisual_t.onClose
// method (which does nothing).

method mainAppWindow_t.onClose;
begin onClose;

    // Tell the winmain main program that it's time to terminate.
    // Note that this message will (ultimately) cause the appTerminate
    // procedure to be called.

    w.PostQuitMessage( 0 );


end onClose;





// When the application begins execution, the following procedure
// is called.  This procedure must create the main
// application window in order to kick off the execution of the
// GUI application:

procedure appStart;
begin appStart;

    push( esi );

    // Create the main application window:
```

```
    mainAppWindow.create_mainAppWindow
    (
        applicationName,        // Window title
        w.WS_EX_CONTROLPARENT,  // Need this to support TAB control selection
        w.WS_OVERLAPPEDWINDOW,  // Style
        NULL,                   // No parent window
        formX,                  // x-coordinate for window.
        formY,                  // y-coordinate for window.
        formW,                  // Width
        formH,                  // Height
        howl.bkgColor_g,        // Background color
        true                    // Make visible on creation
    );
    mov( esi, pmainAppWindow ); // Save pointer to main window object.
    pop( esi );

end appStart;



// appTerminate-
//
//  Called when the application is quitting, giving the app a chance
// to clean up after itself.
//
// Note that this is called *after* the mainAppWindow_t.onClose method
// executes (indeed, mainAppWindow_t.onClose, by posting the quit message,
// is what actually causes the program to begin terminating, which leads
// to the execution of this procedure).

procedure appTerminate;
begin appTerminate;

    // Clean up the main application's form.
    // Note that this will recursively clean up all the widgets on the form.

    mainAppWindow.destroy();

end appTerminate;


// appException-
//
// Gives the application the opportunity to clean up before
// aborting when an unhandled exception comes along:

procedure appException( theException:dword in eax );
begin appException;

    raise( eax );

end appException;



// The main program for a HOWL application must
// call the HowlMainApp procedure.

begin checkBox1;
```

```
    // Set up the background and transparent colors that the
    // form will use when registering the window_t class:

    w.GetSysColor( w.COLOR_MENU );
    mov( eax, howl.bkgColor_g );
    or( $FF00_0000, eax );
    mov( eax, howl.transparent_g );
    w.CreateSolidBrush( howl.bkgColor_g );
    mov( eax, howl.bkgBrush_g );

    // Start the HOWL Framework Main Program:

    HowlMainApp();

    // Delete the brush we created earlier:

    w.DeleteObject( howl.bkgBrush_g );

end checkBox1;
```

The HOWL wCheckable_t, wCheckBox_t, wCheckBoxLT_t, wCheckBox3_t, and wCheckBox3LT_t Classes

Before moving on to the next example, it's worthwhile to quickly discuss the five major classes that HOWL uses to define check boxes.

The `wCheckable_t` class is the main base class for all the check box objects. In HLA v2.9 and earlier, this was an abstract base class; it was promoted to a concrete class in HLA v2.10 and later. Here is the definition of this class:

```
    wCheckable_t:
        class inherits( wButton_t );

            procedure create_wCheckable_t
            (
                wchkName    :string;
                caption     :string;
                style       :dword;
                parent      :dword;
                x           :dword;
                y           :dword;
                width       :dword;
                height      :dword;
                onClick     :widgetProc
            );  external;

            method set_check( state:dword );                external;
            method get_check; @returns( "eax" );            external;

        endclass;
```

The `create_wCheckable_t` procedure is the constructor for the class. The arguments take on the following values:

wchkName: This is a string specifying the object's name. This string should match the identifier (if any) that the application uses for the check box object.
caption: This is the text that Windows will display next to the check box.
style: this is a constant that takes on the following values:

w.BS_CHECKBOX	Creates a small, empty check box with text. By default, the text is displayed to the right of the check box. To display the text to the left of the check box, combine this flag with the w.BS_LEFTTEXT style (or on Windows 95 only, with the equivalent w.BS_RIGHTBUTTON style).
w.BS_AUTOCHECKBOX	Creates a button that is the same as a check box, except that the check state automatically toggles between checked and unchecked each time the user selects the check box.
w.BS_3STATE	Creates a button that is the same as a check box, except that the box can be grayed as well as checked or unchecked. Use the grayed state to show that the state of the check box is not determined.
w.BS_AUTO3STATE	Creates a button that is the same as a three-state check box, except that the box changes its state when the user selects it. The state cycles through checked, grayed, and unchecked.

Possibly OR'd logically with one or more of the following constants (as appropriate:

w.BS_BOTTOM	Places text at the bottom of the button rectangle.
w.BS_LEFTTEXT	Places text on the left side of the radio button or check box when combined with a radio button or check box style. Same as the w.BS_RIGHTBUTTON style.
w.BS_CENTER	Centers text horizontally in the button rectangle.
w.BS_LEFT	Left-justifies the text in the button rectangle. However, if the button is a check box or radio button that does not have the w.BS_RIGHTBUTTON style, the text is left justified on the right side of the check box or radio button.
w.BS_PUSHLIKE	Makes a button (such as a check box, three-state check box, or radio button) look and act like a push button. The button looks raised when it isn't pushed or checked, and sunken when it is pushed or checked.w.BS_RIGHTRight-justifies text in the button rectangle. However, if the button is a check box or radio button that does not have the w.BS_RIGHTBUTTON style, the text is right justified on the right side of the check box or radio button.
w.BS_RIGHTBUTTON	Positions a radio button's circle or a check box's square on the right side of the button rectangle. Same as the w.BS_LEFTTEXT style.
w.BS_TOP	Places text at the top of the button rectangle.
w.BS_VCENTER	Places text in the middle (vertically) of the button rectangle.

parent: This is the handle of the parent window on which the checkbox will be placed. This is typical the main application form's handle, though to could be some other container object on the form if the checkbox is place within a container (such as a `wGroupBox_t` or `window_t` object).
x, y, width, height: This arguments specify the bounding box around the check box and its caption text.
onClick: This is the address of the `onClick` widgetProc event handler. Note that this argument can be NULL, in which case HOWL will ignore clicks on the check box (note that if the

check box is an automatic check box Windows will automatically handling changing the state of the check box; passing NULL as the `onClick` address is common for automatic check boxes).

The `set_check` method sets the internal state of the check box to the value you pass as an argument. The argument's value must be 0/false or 1/true for standard check boxes and 0/false, 1/true, or 2/indeterminate for three-state check boxes.

The `get_check` method returns the current internal state of the check box in the EAX register. This value will always be 0 or 1 for standard check boxes or 0, 1, or 2 for three-state check boxes.

The `wCheckBox_t` class is derived from `wCheckable_t`. This class supplies the `w.BS_CHECKBOX` style (standard checkbox) as the style value for the constructor. That is, it creates standard two-state, left-box, right-text check boxes. Here's the class definition:

```
wCheckBox_t:
    class inherits( wCheckable_t );

        procedure create_wCheckBox
        (
            wcbName     :string;
            caption     :string;
            parent      :dword;
            x           :dword;
            y           :dword;
            width       :dword;
            height      :dword;
            onClick     :widgetProc
        );  external;


    endclass;
```

Other than the absence of the style argument in the constructor call, the constructor's arguments are identical to `wCheckable_t`. Note that this class inherits the get_check and `set_check` methods from the `wCheckable_t` class.

The `wCheckBoxLT_t` class creates a "left text" two-state check box. This form has the check box justified to the right of the bounding rectangle and the text on the left side of the check box. Otherwise, `wCheckBoxLT_t` objects are identical to `wCheckBox_t` objects:

```
wCheckBoxLT_t:
    class inherits( wCheckable_t );

        procedure create_wCheckBoxLT
        (
            wcbltName   :string;
            caption     :string;
            parent      :dword;
            x           :dword;
            y           :dword;
            width       :dword;
            height      :dword;
            onClick     :widgetProc
        );  external;

    endclass;
```

Note that `wCheckBoxLT_t` objects are basically `wCheckable_t` objects with the `w.BS_LEFTTEXT` style.

The `wCheckBox3_t` class implements three-state check boxes. Its declaration is quite similar to the previous two classes (the only difference being the name):

```
wCheckBox3_t:
    class inherits( wCheckable_t );

        procedure create_wCheckBox3
        (
            wcb3Name    :string;
            caption     :string;
            parent      :dword;
            x           :dword;
            y           :dword;
            width       :dword;
            height      :dword;
            onClick     :widgetProc
        );  external;


    endclass;
```

Note that `wCheckBox3_t` objects are `wCheckable_t` objects with the `w.BS_3STATE` style.

The last checkbox type is the `wCheckBox3LT_t` type, which is a combination of the left text and three-state styles:

```
wCheckBox3LT_t:
    class inherits( wCheckable_t );

        procedure create_wCheckBox3LT
        (
            wcb3ltName  :string;
            caption     :string;
            parent      :dword;
            x           :dword;
            y           :dword;
            width       :dword;
            height      :dword;
            onClick     :widgetProc
        );  external;


    endclass;
```

Note that `wCheckBox3LT_t` objects are `wCheckable_t` objects with the (`w.BS_3STATE | w.BS_LEFTTEXT)` style.

Another CheckBox Example: CheckBox2

The *006_checkBox2.hla* application is identical to *005_checkBox1.hla* except it demonstrates a left text check box. There is only one functional difference between the two programs: *006_checkbox2.hla* replaces the `wCheckBox` HDL declaration with the following:

```
wCheckBoxLT
(
    checkBox1,              // Field name in mainWindow object
    "Check Box #1 abcde",   // Caption for push button
    10,                     // x position
    10,                     // y position
    125,                    // width
    25,                     // height
    onClick1                // "on click" event handler
)
```

Checkbox3:

The `007_checkbox3.hla` file extends the previous example by adding a second check box to the form. The declaration is quite straight-forward:

```
wCheckBox
(
    checkBox1,              // Field name in mainWindow object
    "Check Box #1 abcde",   // Caption for push button
    10,                     // x position
    10,                     // y position
    125,                    // width
    25,                     // height
    onClick1                // "on click" event handler
)


wCheckBoxLT
(
    checkBox2,              // Field name in mainWindow object
    "Check Box #2 abcde",   // Caption for push button
    10,                     // x position
    70,                     // y position
    125,                    // width
    25,                     // height
    onClick1                // "on click" event handler
)
```

The interesting thing to note here is that both check boxes (a standard and a left-text check box) use the same on click event handler. Let's take a look, again, at the `onClick1` widgetProc (which hasn't changed in any of these examples):

```
proc onClick1:widgetProc;
begin onClick1;

    // Invert the check in the check box:

    mov( thisPtr, esi );
    (type wCheckBox_t [esi]).get_check();
    xor( 1, eax );
    and( 1, eax );
    (type wCheckBox_t [esi]).set_check( eax );


end onClick1;
```

How does this code figure out whether to invert `checkBox1` or `checkBox2`? Easy. The `thisPtr` value passed into the procedure contains the address of the `checkBox1` or `checkBox2` object. When this widgetProc calls the `get_check` or `set_check` method, it calls that method for the specific check box object passed in the `thisPtr` procedure argument.

One difference between the *checkBox3* and *checkBox2* programs is how the `onDblClick` widgetProc works:

```
proc onDblClick:widgetProc;
begin onDblClick;

    mov( mainAppWindow.checkBox1, esi );
    (type wCheckBox_t [esi]).click();
    mov( mainAppWindow.checkBox2, esi );
    (type wCheckBox_t [esi]).click();

end onDblClick;
```

Notice how this code will simulate a click on both check boxes on the form.

Checkbox4:

The *008_checkBox4.hla* file is an extension of the *007_checkBox3.hla* file that changes the two check box widgets from two-state check boxes to three-state check boxes. In the HDL declaration section (the `wForm..endwForm` statement) this is an almost trivial change; just replace the `wCheckBox_t` and `wCheckBoxLT_t` declarations with the following:

```
    wCheckBox3
    (
        checkBox1,               // Field name in mainWindow object
        "Check Box #1 abcde",    // Caption for push button
        10,                      // x position
        10,                      // y position
        125,                     // width
        25,                     // height
        onClick1                // "on click" event handler
    )


    wCheckBox3LT
    (
        checkBox2,              // Field name in mainWindow object
        "Check Box #2 abcde",   // Caption for push button
        10,                     // x position
        70,                     // y position
        125,                    // width
        25,                     // height
        onClick1                // "on click" event handler
    )
```

Because these are three-state check boxes, we need to modify the `onClick1` widgetProc (from the earlier examples) to handle three states. The following code toggles the check boxes between the three states each time the user clicks on one of the check boxes:

```
proc onClick1:widgetProc;
begin onClick1;

    // Invert the check in the check box:

    mov( thisPtr, esi );
    (type wCheckBox_t [esi]).get_check();
    add( 1, eax );
    if( eax > 2 ) then

        xor( eax, eax );

    endif;
    (type wCheckBox_t [esi]).set_check( eax );


end onClick1;
```

wCheckable_t Declarations

In HLA v2.10 and later, the `wCheckable_t` type was promoted from an abstract type to a concrete type (and an appropriate `wCheckable` statement was added to the HDL). The `wCheckable` declaration takes the following form:

```
    wCheckable
    (
        checkableID,            // Field name in mainWindow object
        "caption string",       // Caption for check box
        style,                  // Check box style
        x,                      // x position
        y,                      // y position
        w,                      // width
        h ,                     // height
        onClickHandler          // "on click" event handler
    )
```

The most interesting part (and the obvious difference from the other check box declarations) is the `style` argument. This lets you specify one of the `w.BS_*` style constants listed earlier in this document. In particular, by using `wCheckable`, you can create `w.BS_AUTOCHECKBOX` check boxes and `w.BS_AUTO3STATE` check boxes. These automatic check boxes will handle user clicks without you writing an on-click event handler. Just supply a NULL address and Windows will take care of the rest for you.