Custom Windows/Dialogs
(C)1994 Spotware
by Neil St.Laurent
Version 2.00

References

(No, I never forgot the page reference or something of the tpye, if you can't figure it out then you may not want to use these classes, oh, okay, for those of you still new with online text you will note that searching for the topic in the table of contents should take you to where you want to go.

..Introduction..

This collection of custom window and dialog classes were created because I felt that standard windows as getting too boring, maybe to the extent on infringing on my rights to express things as I please. Anyways, those folks at Microsoft made it annoying to make your own window border and title bar, there are a bunch of NC_ messages, but they require a bit of extra work to use them. These classes are meant to be used with a BORLAND Windows C++ compiler, I created them with Turbo C++ for Windows. They should make it fairly simple to draw in the border, they take care of all the "behind the scenes" control. It took me a long time to create these classes, there were a lot of errors that are hard to notice at first glance. For all of you that will try to create your own classes for custom windows, here are the errors that I encountered that you might also encounter:
- Windows draws its the normal border on top of yours, ignoring your attempt to change it.
- The window doesn't redraw itself when it should.
- Active and Inactive states get mixed up or aren't reflected, even if you took them into consideration.
- The window takes over the system and doesn't let you switch anywhere else.
- When the window is moved the windows behind it don't update themselves.
- When the window is closed the windows behind it don't update themselves.
- The border gets redundantly drawn slowing down the system.
- When the window is close it occassionally draws a final frame leaving a ghost window on the screen.

These classes are of course, not free. You may experiment with them all you want, but if you ever want to use them in a program that will be distributed in any manner then you must pay a fee for their use, more information is further on in this file.

I included one example program so you can see how to use the classes. There is now more code in the package, another project for custom controls.

One final note, I hope I didn't miss anything in these brief explanations, please write or call, or get a hold of me on Idiot Box or other BBS and network systems. I will try to help whenever possible, if required writing some code specific to your problem. Please try other Spotware programs, which on the data of this release have all be written by myself, I am growing a little more in my programming and will be creating more REAL programs and possibly even APPLICATIONS rather than many small utilities. Please write for a list of Spotware products, if you care at all.

..DLLS..

..Note..

Make sure that the DLL is accessible by the program, either in the program path a path specified in the path environment variable, or in the windows directory. The three BORLANDS DLLs are also required, those are the Run-Time, Container Class, and OWL.. The Custom Windows / Dialogs library is also provided in .LIB form in the Large class to be linked in with your program. When using the .LIB form you may want to load the library CUSTRES.DLL to have access to all the resources that the CUSTWIND.DLL uses. When using static linking the instance members of CustColors and CustCursors, if not using windows defaults, must be set. Please see ..Functions.. for more info.

..Resources..

There are many resources in the Library DLL which are duplicated in the pure resource

DLL for use with static linked programs. Here is a list of the ones you may need to use, either for the cursors, the background of TBitOver ro for NCButtons, there are many others but you probably have no use for them:

BITMAPS

BACK_BRICK	Bricks
BACK_EXCLAMATION	! used by CMessageBox
BACK_FISH	Bright Fish
BACK_INFORMATION	i used by CMessageBox
BACK_QUESTION	? used by CMessageBox
BACK_SPOTWARE	Spotware Logo
BACK_STOP	Stop Sign used by CMessageBox

CHK_BACK	AND Mask used by NCRadioButton class
CHK_?LIGHT	For NCRadioButton reflects a lighted state
CHK_?OFF	For NCRadioButton reflects a darkened state

One of the following colors should replace the ?:
BLACK, BLUE, CYAN, GREEN, PURPLE, RED, YELLOW

CUSORS

IDC_DIAGLEFTBORD	Diagonal Left Border
IDC_DIAGRIGHTBORD	Diagonal Right Border
IDC_HORIZBORD	Horizontal Border
IDC_MAIN_CUR	Main Cusor, impossible geometric shape
IDC_TITLEBAR	Title Bar
IDC_SYSMENU	System Menu
IDC_VERTBORD	Vertical Border
IDC_MENU	Menu
IDC_MAXIMIZE	Maximize Button
IDC_MINIMIZE	Minimize Button
IDC_RESTORE	Restore Button
IDC_BORDER	Non-Sizeable Border
IDC_VERTSCROLL	Vertical Scrollbar
IDC_HORIZSCROLL	Horizontal Scrollbar
IDC_NCBUTTON	NC Button

..CLASSES..

There are many classes in version 2.00, it depends on how ambitious you get that determines which ones you use, there are also structures not listed here:

..TUnderlying..

This Tree diagram looks like hell and if you can follow the inheritence you are already have a good understanding of how the classes work. This whole mess is caused by the fact that DDVT's cannot be multiply inherited, and that TDialog is not derived from TWindow, if either of these were so the entire thing may look nicer.

```
TUnderlying------------------------------\
        |                                |
        |---TBitOver                     |----------------------------------\
        |---TCustomDialog                \--TBarWindows\---TBlockWindows
        |         |                            |         \                |         \
        |         |------------------------------TBarDialog |             |          |
        |         \---------------------------------------------+---------------TBlockDialog |
        |                                              |                           |
        \---TCustomWindow                              |                           |
|                                       |                         |
|---------------------------------------TBarWindow                |
\-----------------------------------------------------------------TBlockWindow
```

TBarDialog	Bar Dialog Class for use with dialog boxes
TBarWindow	Bar Window Class for use with windows
TBarWindows	Bar Windows base class
TBitOver	Class the tiles bitmaps in client area
TBlockDialog	Block Window class for use with dialogs
TBlockWindow	Block Window class for use with windows
TBlockWindows	Block Windows base class
TCustomDialog	Custom Dialog Class for use with dialog boxes
TCustomWindow	Custom Window Class for windows
TUnderlying	Basic Behaviour and Defaults for Custom Windows

The derived classes each have a unifying base class so that the functions to customize need only be defined once then incorporated into both the Dialog and Window class. See the example BLOCKS.H header file to see how this is done, I would explain it but it is done the same way every time and if you really want to know you could figure it out.

Many of the classes require new parameters, I mean ones not found in regular OWL or Windows. Here is a list of constructors and their parameters, each won't be described over and over, only the new parameters will be:

TUnderlying(PCustColors Colors, PCustCursors Cursors)

Colors	The set of colors to be used, takes a copy of the structure.
Cursors	The set of cursors to be used, takes a copy of the structure.

TBitOver()

See Contents Reference for its use

```
TBarWindows(PCustColors SomeColors, PCustCursors SomeCursors)
TBarDialog(PTWindowsObject AParent, LPSTR AName, PCustColors Colors,
        PCustCursors Cursors, PTModule AModule = NULL)
TBarDialog(PTWindowsObject AParent, int ResourceId, PCustColors Colors,
        PCustCursors Cursors, PTModule AModule = NULL)
TBlockWindows(PCustColors SomeColors, PCustCursors SomeCursors)
TBlockDialog(PTWindowsObject AParent, LPSTR AName, PCustColors Colors,
        PCustCursors Cursors, PTModule AModule = NULL)
TBlockDialog(PTWindowsObject AParent, int ResourceId, PCustColors Colors,
        PCustCursors Cursors, PTModule AModule = NULL)
TCustomDialog(PTWindowsObject AParent, LPSTR AName, PCustColors Colors,
        PCustCursors Cursors, PTModule AModule = NULL)
TCustomDialog(PTWindowsObject AParent, int ResourceId, PCustColors Colors,
        PCustCursors Cursors, PTModule AModule = NULL)
```

AParent	Pointer to the parent OWL object.
AName	Name of resource of the dialog template.
ResourceId	ID of resource of the dialog template.
AModule	Module to find template resource

..TNCButton..

TNCButton

```
|
|---TNCBitButton
|          |
|          \---TNCSystemButton
|
|---TNCCharButton
\---TNCRadioButton
```

TNCButton	Base Class for NC buttons
TNCBitButton	Bitmap NC Button
TNCCharButton	String NC Button
TNCRadioButton	Radio NC Button
TNCSystemButton	Bitmap NC Button that sends a system message

TNCButton(HWND AParent, RECT& AWindow, int AnId)

AParent	Handle of window to send messages to
AWindow	Bounding rectangle that the button is contained in
AnId	Identifier to use when sending messages

TNCBitButton(HWND AParent, RECT& AWindow, int AnId, HINSTANCE Inst, LPSTR BitUp, LPSTR BitDown)

Inst	Instance to locate bitmap resources
BitUp	Reference to bitmap resource of button in up position
BitDown	Reference to bitmap resource or button in down position

TNCCharButton(HWND AParent, RECT& AWindow, int AnId, HINSTANCE AnInst, char* AChar)

AnInst	Instance to the CustomWindow resources. See ..Functions..
AChar	Pointer to a string to use, copies the string.

TNCRadioButton(HWND AParent, RECT& AWindow, int AnId, HINSTANCE AnInst, LPSTR On, LPSTR Off, char Change, char IState)

AnInst	Instance to where the On and Off state bitmaps can be found
On	Resource reference to Bitmap of button when it is on
Off	Resource reference to Bitmap of button when it is off
Change	TRUE/FALSE can radio button be changed by the user with the mouse
IState	Initial state of radio button. See the NCBUTTON.H file.

TNCSystemButton(HWND AParent, RECT& AWindow, int Type, HINSTANCE Inst, LPSTR BitUp, LPSTR BitDown)

Type	Type of message to send, refer to WM_SYSCOMMAND message.

..CustColors..

int	Integer Members
Size	Size of structure, used for future compatibility. Version 1.00 has a size of 354

COLORREF	COLORREF Members (Used to set the color of areas of the Window)
TitleBarActive	Color of active title bar
TitleBarInactive	Color of inactive title bar
WindowBack	Background of Window
TitleTextActive	Color of active title bar text

TitleTextInactive	Color of inactive title bat text
UpLeftBorders	Color of Top and Left borders of window.
RightBottomBorders	Color of Bottom and Right borders of window
ButtonText	Color of text in a button
ListBoxText	Color of text in a list box
StaticText	Color of static control text
EditText	Color of edit control text

Public Member Functions

void WindowsDefaults(void)	Sets normal windows colors, set by user.
void OrigClassDefaults(void)	Sets the colors to the ones that I used when I first create the class.

..CustCursors..

(Used to define the cursors to be used in certain areas of the window)
(The various sections are the resource id of the cursor)

int	Integer Member
Size	Size of structure for future compatibility. Version 2.00 Size of 32
VerticalBorder	Vertical Border of left and right.
HorizontalBorder	Horizontal Border on top and bottom.
DiagonalLeftBorder	Upper Left and Bottom Right corners.
DiagonalRightBorder	Bottom Left and Upper Right corners.
Main	Uncovered portion of client area.
TitleBar	Title Bar.
SysMenu	System Menu.
HorizontalScrollBar	Horizontal Scroll Bar.
VerticalScrollBar	Vertical Scroll Bar.
Menu	Menu Bar and popup menus.
Maximize	Maximize Button.
Minimize	Minimize Button.
Border	Non-sizeable border.
NCButton	NC Button

HINSTANCE	HINSTANCE Members
Instance	The instance where the cursors are located.

Member Functions

void WindowsDefaults(void)	Uses all the windows default cursors.
void OrigClassDefaults(void)	Uses the original class cursors, you must specify the instance of the DLL in the Instance member, this is set automatically by the function.

..TBitOver..

This is one of those new features and helps create more of a custom window. It tiles the background of the client area with a supplied bitmap. The constructor takes no arguements, this rids you of some over problems that come up when it does, but anyways you have to call one of two functions before the setupwindow function is called, usually in the constructor of the derived class, they are below with the other functions.

void SetBitmap(HINSTANCE Inst, LPSTR BitName)

Inst	Where the bitmap can be found
BitName	Name of the resource

This function sets the class to destroy the loaded bitmap when the class is deleted.

void SetBitmap(HBITMAP ABit)

ABit	A handle to an existing bitmap.

This function sets the class so it won't destroy the bitmap when the class is deleted.

void TurnOnDestroy(void)

Sets the class to destroy the bitmap it has when the class is deleted.

void TurnOffDestroy(void)

Sets the class to not destroy the bitmap it has when the class is deleted.

Well, those were the classes, they are fairly easy to figure out how to use, you should by now know how to create a window with the original class defaults, that is without redifning certain member functions to redraw the window. The more custom you want the classes the more work you have to do, the best way to understand how to do this is to look through the CUSTXAMP project and BLOCKS source code.

..Important Note for 1.00 Users..

The call to the TUnderlying function constructor is not longer automatically made by any of the base classes, each new derived class must explicity call the constructor passing a CustColors and CustCursors pointers. This is because to allow multiple inheritance for easy creation of new Window and Dialog classes the TUnderlying was declared as virtual with many calls to it, even more with TBitOver added in, so the final call must be made by the derived class.

Example

```
class TNewWindow : public TCustomWindow
{
public:
        TNewWindow(PTWindowsObject AParent, LPSTR ATitle, PCustColors SomeColors,
                PCustCursors SomeCursors, PTModule AModule = NULL)
                : TCustomWindow(AParent, ATitle, SomeColors, SomeCursors),
                TUnderlying(SomeColors, SomeCursors)
                { };
};
```

..Custom Level..

The custom level of the window can be changed. The following are the functions that are used to set the custom level:

void SetNoCustom(void)

Only uses custom cursors, the window is left to be drawn with the Windows defaults.

void SetLightCustom(void)

Does not draw in the window frame, it still sets the proper colors in the client area and uses the custom cursors.

void SetHeavyCustom(void)

Uses all the custom colors, custom cursors, draws in the non-client area and sets the colors in the client area. This is the default and I strongly recommend only using it, without this mode set none of the NC buttons will work properly, including the fixed dialog minimize button. In version 2.00 I left them in only as a last resort to speed up the computer, but they truly don't help if a class was put together properly, the Bar classes work incredibly quick and if you are using TBitOver you can not use it to get it going faster. Also many NC Buttons slow it down slightly. If you really want to use windows defaults you can call the windows default process of NCPaint in the UserFrame function and not define the other custom drawings.

These functions are very useful sometimes. If the person has a slow computer you may want to let them turn them off so it won't go too slow, or you may just feel like giving the person an option as to whether you use the windows defaults or your new windows. I personally feel that I spent a lot of time working on these functions and I didn't waste it so somebody can look at the default ones, they express your creativity, remember that.

..Re-Paint..

Before you start repainting you may want to know that there are various data members in the class filled out that derived classes can use, they are declared as protected. They are as follows:

..Member Variables..

HBRUSH Background	Brush used to paint the background.
struct Chars	Bit Fields Structure
IsActive	TRUE=Window is Active FALSE=Window is InActive
Closing	TRUE=Window is Closing
SysMenu	TRUE=Window has a system menu
MinBox	TRUE=Window has a minimize box
MaxBox	TRUE=Window has a maximize box
Caption	TRUE=Window has a caption
CustomLevel	0=Windows Default Window/Dialog 1=No custom border, but colors are still custom 2=Full Custom Level, borders, cursors, and all
IsChild	TRUE=Window is a child window

RECT   WindowRect   Bounding rectangle for the entire window. The left and top of this structure are always 0, and the remaining rectangles and coordinates are all relative to the upper left of the window, that is the WindowRect.

RECT TitleRect	Bounding rectangle for the title area.
RECT ClientRect	Bounding rectangel for the client area.
POINT SysMenu	Coordinates for where System Menu should be drawn
POINT MaxButton	Coordinates for where Maximize Button should be drawn.
POINT MinButton	Coordinates for where Minimize Button should be drawn.

int BorderSize	Thickness of the Border.
int BitWidth	Width of Bitmaps in titlebar.
int BitHeight	Height of bitmaps in titlebar.
int TitleHeight	Height of the titlebar.

CustColors DrawColors	Custom Colors supplied on creation.
CustCursors MouseCursors	Custom Cursors references supplied on creation.

..Paint Functions..

Those member variables should make your job of creating custom drawn windows a lot easier, most of the things needed are already figured out for you.

The following are the member functions to be redifined, any one not redifined will just use the original class defaults:

virtual void UserFrame(HDC DrawDC)

This function should be redifined if you want to draw your own frame, this is limited to just the border, excluding the title bar area.

virtual void UserTitle(HDC DrawDC)

This function should be redifined if you want to draw your own title bar, excluding the system menu and buttons. Remember to use the member variables to decide whether the window is active so you can use the appropriate color. Don't forget to write the window title in the title bar.

virtual void DrawSysMenu(HDC DrawDC)

This function should be redifined if you don't like Windows default system menu bitmap, here you can draw whatever you feel like, you don't have to worry about it when it is pushed because windows just inverts the colors. You can draw a happy face or whatever.

virtual void DrawMinMax(HDC DrawDC)

This function should not be redifined because there are no provisions for drawing the right picuture when the button is pressed. In future version this will be supported properly. I guess I lied, this is never going to be properly supported, if you need different versions of these buttons remove them from the style of the window and add two TNCSystemButtons. You need to derive a new class for the maximize button because it is reflected differently in normal and maximized windows, and sends different messages, will probably be in a future version.

There is now and example of how this is done, the BLOCKS.H and BLOCKS.CPP define a class that creates a new border, even a new client area.

..Functions..

Here are some of the miscellaneous functions available in the DLL and classes, I don't show a complete list anywhere because you can look in the header file for that, many of the functions you just won't use, and it is easier to look up what you want to do than look at every function and see what it does, but here are the unusual ones.

In the DLL:

HINSTANCE GetCustWindInstance(void);

Returns the instance of the DLL

PTModule GetCustWindModule(void);

Returns a pointer to the module class of the DLL.

int CMessageBox(PTWindowsObject Parent, LPCSTR Text, LPCSTR Title, UINT Style, PTModule Module)   (EXTRAS.H)

Parent	Parent Object
Text	Text to display.
Title	Title to display.
Style	See MessageBox in API reference for styles
Module	Module where to find dialog template, the customwindow resources.

..NCButtons..

NC Buttons are one of the bigger parts of the new features, this section describes the public member functions in TUnderling, and all derived classes, that handle the use of NCButtons.

1st Note: When a button is pressed it sends and ID message to the sepecified window using the buttons id, therefore you can use the DDVT features and regular processing messages to deal with the buttons. The Ids need not be unique with other controls, so you may have an NC button and regular button with the same ID and therefore sending the exact same message.

There is a maximum of 20 NC Buttons per window, that is way more than enough so you should have no problems with it, if you do then please write to me and I will help you find a way around or increase the number, or make it so there is no real maximum.

Here are the functions and what they do:

void DrawNCButton(int Button)

Forces the redraw of the button with the id of Button. This should be called whenever the program changes the button, usually when the radio button is changed. Specifying -1 as an Id will redraw all the buttons.

BOOL AddNCButton(PTNCButton AButton)

Adds a button to current ones with the given pointer to the button, the passed parameter may be any button dervied from the base class TNCButton. TRUE/FALSE is returned to indicate if the button was added, if FALSE is returned it means there are already 20 buttons.

BOOL RemoveNCButton(PTNCButton AButton)

Removes the first occurance of the passed button, I say this because if you added a button more than once, by accident or something, the first one will be removed. These need not be called before destruction. This does not delete the memory associated with the button.

BOOL RemoveNCButton(int Id)

Given an Id this function removes the button from the list of buttons and deletes the memory associated with the class.

PTNCButton GetNCButton(int Id)

Returns a pointer to the button with the passed Id.

void AddExtendedButtons(HINSTANCE Inst, int StartRight)

Inst	Instance where resources can be found, the CustomWindow resources.
StartRight	Button Position from the right where to start putting the buttons.

Adds the extended buttons into the NC button list, which include the ScreenSaver and TaskList buttons. Before the class is destroyed you must call the DeleteExtendedButtons function or else valueable resources will be not be freed. This function should be called during SetupWindow.

void DeleteExtendedButtons(void)

Should be called in the destructor of a class if the AddExtendedButtons was called.

void PosFromTopLeft(RECT& ARect, int APos)

Given a reference to a a rectangle and APos in button terms from the left this function wil fill in the ARect structure with the coordinates and dimensions of the button in that position.

void PosFromTopRight(RECT& ARect, int APos)

Given a reference to a a rectangle and APos in button terms from the right this function wil fill in the ARect structure with the coordinates and dimensions of the button in that position.

These buttons can really add a lot to a program, who needs a program wide action bar or whatever that wastes space when every window can have its own set of buttons that don't use up any extra space. If more than one window uses the same buttons and send messages to the same window you can use the exact same button (as I did with the SysSound program).

..File List..

Here are the files that should be with this package. If you redistribute the package it must have all the unmodified original files in it, if the following files are not in the package you got (either from a download or otherwise) you should notify the person who uploaded it, if you want all the files you can always get them (and the newest version) from the Idiot Box RBBS-PC or send me $5 to cover the cost of a disk and shipping and I will mail them out.

CUSTWIND.WRI	This file.
BAR.H	Header file for TBarWindows Classes
BITOVER.H	Header file for TBitOver Classes
CUSTID.H	Identifiers for certain resources
CUSTL.LIB	Custom Windows/Dialogs Static Link Lib Large Module
CUSTRES.DLL	Resources DLL
CUSTSTRC.H	Header file for structute definitions.
CUSTWIND.H	Header file for class definitions.
CSTWND20.LIB	Library file to use the DLL.
CSTWND20.DLL	Custom Windows/Dialogs Dynamic Link LibraryVer 2.00
NCBUTTON.H	Header for NC Button Classes
CUSTXAMP.CPP	An Example Program
CUSTXAMP.PRJ	Project File for The Example.
CUSTXAMP.RC	Resource Script for eaxmple
CUSTXAMP.EXE	Executable of the Custom Example
XAMPIDS.H	Ids for controls in example.
BLOCKS.H	Example of redefining border drawing an client area
BLOCKS.CPP	Example

CUSTCTL.CPP	Main Source for Custon Control DLL
CUSTCTL.DEF	Def file for Custom Control DLL
CUSTCTL.DLL	DLL for Custom Controls
CUSTCTL.H	Header for Custom Control DLL
CUSTCTL.LIB	Library for exported functions in DLL
CUSTCTL.PRJ	Project File for Custom Control DLL
CUSTCTL.RC	Resources for Custom Control DLL
NRADIO.H	Header for Custom Radio Button
NRADIO.CPP	Custom Radio Button
STATEBUT.H	Header for a 3-State Button
STATEBUT.CPP	Three State Button Base Class
CCHECK.H	Header for Custom Check Button
CCHECK.CPP	Custom Check Button
EXTRAS.H	Header file for various functions

..New Features..

Here are the new features since version 1.00:

-Decrease in flickering and useless border redraw
-Fixed many repaint problems with the client area and border
-Border drawing for TUnderlying border draw has been sped up
-Push Buttons and Radio Buttons in the title bar
    -Fixes the misplaced minimize button problem with dialog boxes by using an NCButton (now requires that CustomLevel is 2 otherwise it won't work)
    -Extended buttons to press for the task list and screen saver
    -String Buttons
    -Bitmap Buttons
-Example of redrawing the frame and Client area with the Blocks class
-TBarWindows class for a new style border and title bar, works much faster than the TUndrelying defaults so may be preferable to the defaults
-TBitOver class to make the client area background a tiled bitmap
-Regions calculated more accurately (may make your program look a little different)
-CMessageBox which is a custom window equivalent of the MessageBox API call
-Two custom controls, radio and checkbox that differ from the Windows ones because the text does not draw it's own background and the bitmaps update themselves properly to allows a tiled bitmap client area show through

Users of 1.00 should skim this file to see how the new changes effect your old code, one big not compatible change is the dialog box minimize button optimization, for a dialog box to have a minimize button requires if custom level be at 2, nothing lower otherwise the button will not get drawn. Sorry, but if you really want to support the custom levels you are depriving yourself of showing others your creativity.

..Addresses..

SPOTWARE
Box 1064
Hanna, AB, T0J 1P0
Voice: (403) 854 - 3860

Neil St.Laurent (Can reached at Spotware, just put my name on the envelope).

..Registering..

Didn't think I was going to skip this part did you? Well, like I said, I put a lot of work into this and I am still continuing my work on more custom window routines (can't tell what they are yet because somebody else may make them). With registration you are allowed to distribute programs that use the Custom Windows/Dialogs classes and you will get the newest version of Custom Windows/Dialogs (with your name as part of the list of people/companies registered to use it. Registation for Version 1.00 is $50, you will also get a list of programs available from Spotware. Use the form below for ordering:

<u>**Custom Windows/Dialogs Registration**</u>
Complete and mail to:
Spotware
Box 1064
Hanna, AB, T0J 1P0
Attn: Custom Wins/Dlgs

Company:
Contact / Name:
Mailing Address:
Street Address:
City:
Province / State:
Country:
Postal / Zip Code:

Disk Size:	3.5"	5.25"

Version 1.00 of Custom Windows/Dialogs	$50
Shipping And Handling	$5

Total	$55

Money order for $55 (US Funds) is enclosed. (Yes) / No
(Yes is circled because if it isn't enclosed then you aren't ordering and shouldn't be using this form. Money order is the only payment accepted)

Types of Programs it will be used with:

Comments / Suggestions: