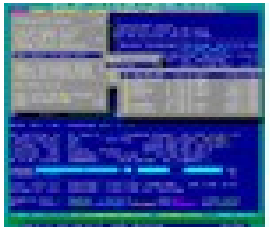
TxWin 5.xx

Programming and User Guide

Jan van Wijk

Brief programming and user guide for the open-source TxWin text UI library

FSYS - Software



TxWin

Presentation contents

- Interfacing, include files, LIBs
- The message event model
- Structure of a typical application
- Setup and creation of windows

- Window classes
- Window procedures and messages
- Window handling functions
- Standard dialogs
- Command line and output buffer usage
- The menu system
- Using functional TRACE
- Miscellaneous functions

Interfacing to TxWindows (TxWin)

- To use TxWin from a 'C' application all you need to do is include the 'txlib.h' header file. In turn this will include other TxWin stuff as well as some common 'C'-library and operating system specific headers.
- Compile and link with the TxWin library ('*.LIB' or 'lib*.a' file) that matches the platform and the variant you want (retail, debug).
  - This is part of the standard master makefile (.MIF or .OSX)
- TxWin comes as STATIC libraries (.LIB or .a) not dynamically loaded ones (.DLL or .so)

The message event model

- TxWin uses almost the same model as PM/WIN:
  - Windows are created and inserted in a Windows hierarchy, each window has a unique HANDLE
  - Communication with (and between) windows is mainly done using messages (events) that are addressed to its handle, relating to the windows procedure that should process the message
  - This message based system allows for a very modular distribution of functionality and easy changes to appearance or behaviour without having access to the base window class code (sub-classing).

Message handling

- Messages can be 'sent' directly to a window for synchronious execution of the related code (much like a function-call) or 'posted' to a QUEUE with normal processing continueing.
- After finishing current processing, the OLDEST message will be picked up from the queue, and sent to the window it was addressed to.
  This 'dispatching' is done by the message loop either in the main-program or inside a dialog.
- When the queue is empty, it waits for new events, usually keyboard or mouse ...

What is a (TxWin) window ?

- An object that defines a text-area with a number of lines and columns, with associated behaviour
- On the screen: a rectangular area showing the window frame plus contents. The window can be invisible or partly covered by other windows.
- In the program: a data structure that holds all information about the window
  - For many window classes, there is also a link to the contents of the window, usually as a pointer to a data structure and some descriptive fields.

TxWin versus PM/WIN windows

- When you are familiar with PM/WIN windowing it is good to realize a few major differences:
  - TxWin is purely TEXT based, no graphics are possible, limiting appearance freedom and amount of information that can be displayed and handled in one window
  - A Window in TxWin includes borders, a title and footer, special areas like close-buttons and a client-area.
    In PM/WIN all of these are SEPARATE windows!
  - Access to (contents) data is usually direct, with variables instead of through a purely message based interface as many PM/Windows classes use.

Application structure

- In addition to standard 'C' stuff like main:
  - Initialize the library, including argument handling
  - Interpret command line switches, if any
  - Create and initialize the main desktop window
  - Create one or more application windows
  - Start the action: Show a window or post a MSG
  - Enter the main message-loop (event dispatch)
  - Handle messages (events) in window-procedure
  - On exit, terminate library and cleanup

Creating Windows

- All windows and dialogs in TxWin are created dynamically, there are no static resources like OS/2 PM or Windows has.
  - Macros and widgets can streamline this ...
- Typical window creation consists of:
  - Initialize window-setup structure, directly or using txwSetupWindowData(...)
  - Create the window using txwCreateWindow(...)
  - Add or attach the window contents, depending on the class of the window (text, lists, etc ..)

Window classes

- Window classes define TYPES of windows
- The specific appearance and behaviour of a class is implemented in the library, mainly in the form of the default window procedure that handles all standard messages for the class like painting and user input
- Applications can add to or change this by using a specific window procedure (sub-classing) on a per-window basis (not the class), where multiple windows may share the same window procedure

Window procedures

- A window procedure is a function being called for every message sent to a specific window.
- The structure is simple, select (switch) on the message-id, and perform required actions.
- The window procedure handles any messages it is interested in, and passes all others to the default one: txwDefWindowProc()
- Specific windows procedures can be assigned to any window on creation to allow changing the appearance or behaviour

Sub-classing windows

- Changes appearance and/or behaviour
- Unlike PM/WIN, subclassing in TxWin works on a per-window basis, and NOT a whole class.
- Implemented by allowing a window-procedure to be defined on the txwCreateWindow() and a few other related functions like txwDlgBox()
- There is no need to register new classes, you use the existing ones and add window procedures where needed

TXW_FRAME class

- This is the simplest of classes:
  - Has an optional BORDER with title and/or footer
  - Does NO painting of the client area (transparent)
  - Can save underlying contents, and restore on destroy
  - No user data is associated with this window
- It is rarely used 'as is' but serves as the main 'desktop' window and as main window for a dialog completely covered by its 'controls'.

TXW_CANVAS class

- This is a frame, with a default 'client area':
  - Client area can be filled with a solid color
  - Optional 'ASCII artwork' can be defined to appear in the client area (see TXT test program)
- Often used as main window for a dialog. The client area then forms the 'empty' areas between the dialog control windows like buttons, entry fields and lists.

TXW_STATIC class

- This is a simple, multi-line text area to hold static text for display (NOT editable!)
- The data is a standard TXWin array of string-pointers allowing use of either statically defined as well as dynamically created texts. (char *text[])
- Useful to add descriptive texts to dialogs

TXW_STLINE class

- This is an even simpler, single-line text area to hold static text for display (NOT editable!)
- The data is a standard 'C' string-pointer allowing use of either statically defined or dynamically created texts (char *text)
- Useful to add short descriptive texts to dialogs, like header lines for tables or entry fields
    - Note that for entry field headers you can use the 'title' from the border area as well

TXW_SBVIEW class

- This is a multi-line, output only text window with several special properties:
  - New text can be added to the end of the buffer using the standard TxPrint() function (printf like interface)
  - ANSI style colors and some positioning can be used
  - The displayed text is kept in a large 'scroll-buffer', colors are preserved when scrolling back and forth
  - Display of the buffer is optimized to allow smooth scrolling even when other windows are displayed on top of it (including shadowing :-)

TXW_SBVIEW, continued

- Footer line of the window has automatic line counters
- Application can display short status messages to that footer line as well, like progress indicators

- It is the standard **OUTPUT WINDOW** when producing a lot of loosely structured information

- Note: Similar to showing status messages in the footer of the scroll buffer, applications can display messages in the desktop title (or top) line as well.

  This is also used by the functional trace facility (toggled using the Alt+/ key)

TXW_ENTRYFIELD class

- This is a single-line text entry field with some editing capabilities
- The data is a standard 'C' character array.
  (char *text)
- An optional history-buffer can be attached to automatically store entered values for later retrieval, either using the arrow-keys or a popup-list. Useful for a command line :-)

TXW_TEXTVIEW class

- This is a simple output only view window for text
- Text may be (much) larger than the window
- Scrolling through the text possible using the arrow-keys, PgUp/PgDn etc, as well as some controls for the mouse in the border (if any)
- When available, the mouse scroll-wheel can be used to scroll up and down too
- Is used internally for displaying HELP screens

TXW_BUTTON class

- This is a button control, in the form of:
  - **Push** button, where a click or ENTER on the button performs some kind of action like [OK], [Cancel] etc
  - **Radio** button, usually several in a group where only one button in the group has the 'ON' status.
    Clicking on a button inverses the state of that button, and possibly all others in the group
  - **Check** button, either single or in a group, where each button has its own checked/unchecked status.
    Clicking the button inverses the state but does not affect any other button

TXW_BUTTON, continued

- The data is a simple BOOLEAN variable for the Radio and Check buttons, and there is no associated data for a push button.
  - Click events are communicated using messages
  - Radio and Check buttons can have the 'AUTO' property, meaning that all handling for the button is done entirely by the library. All the application needs to do is read the boolean variable ...
  - Grouped buttons (or any grouped controls) are seen as a single entity for TAB-key navigation. In this case the arrow-keys still navigate within the group ...

TXW_LISTBOX class

- This is a generic LIST control in the form off:
  - Singe selection list, like a (menu) popup
  - Multiple selection list, where more than on item in the list can be marked as being 'selected'
- The data is an array of 'TXSELIST' items that can either be statically defined (for the main menu for example) or created on the fly (like for directory lists)
  - There can be more items in the list than visible, and scrolling is supported in as with the regular text view
  - Items can be marked as 'disabled' or 'separator'

TXW_LISTBOX, continued

- Lists can have several visual appearances:
  - As a popup-window, like a menu or floating popup
  - Embedded as a control in a dialog, like the directory and file-lists in standard file dialogs
  - As a 'spin' control with a single (current) value visible and the arrow-keys 'spinning' through all available values. The ENTER key will present a popup version. An example is the drive selection list in file dialogs.

TXW_HEXED class

- This implements a complete HEX-editor on one or more (up to 9) memory buffers
- Includes a HEXadecimal and ASCII area, both are editable (and update the other area :)
- Reading and writing is delegated to the calling application using a callback function, to use it as a disk/sector editor or edit various other data
- Supports a variable number of (byte) columns and rows, to match the data structure

Dialog windows

- A dialog consists of a base-window, usually a CANVAS class, and one or more *CONTROLS*
- Each control is a window of its own, and can be any of the presented classes.
- Dialogs are built dynamically by creating the dialog-frame and all control windows.
  It is then presented using txwDlgBox()
- Dialogs are MODAL in nature, meaning that other parts of the application are NOT operational while the dialog is up.

Dialog windows, continued

- Specific window procedures can be defined for the whole dialog and/or each control
  (or the default txwDefDlgProc() / txwDefWindowProc() are used)
  - Data is made available to the dialog using the attached data for each control.
  - In addition to that, a data structure could be attached to the frame-window (window-ptr) to be used by the dialog window procedure
  - The return code identifies the control (button) that was active when the dialog was ended
  - Possibly modified data stays available after return

Dialog Widgets

- Widgets are normal controls combined in a *WIDGET-ARRAY* that is processed in one go
- It is an easy way to define complex dialogs, without having to create each window separately
- There is a dedicated (empty) canvas dialog that will display and handle such a widget-array
- Many TxWin standard dialogs will accept a widget array as an optional parameter, and add the widgets to that dialog

  FileOpen, FileSaveAs, Message, Confirm and more ...

Standard dialogs

- TxWin includes a few standard dialogs:
  - Message-box, with one to four buttons (W*)
  - Prompt-box, to get simple single field input (W*)
  - File-Open and File-save-as dialogs (W*)
  - Directory picker dialog (W*)
  - Menubar dialog, with pulldown and sub menus
  - List-box, as submenu or standalone popup
  - Widget dialog, easy creation of custom dialogs (W*)
- The (W*) marked dialogs can be extended very easily using an array of Widget definitions
  - Any CONTROL class can be used in a widget list

More info

- This document is NOT a reference, and actually there IS no reference at the moment :-) So:
  - Get the TxWindows library, download available from:

    https://www.dfsee.com/download/#txwin

  - Check out the samples and TXT test application
  - Really READ the available HELP screens :-)
  - Study the interfaces as defined in the header files
  - Study behaviour by looking through the sources

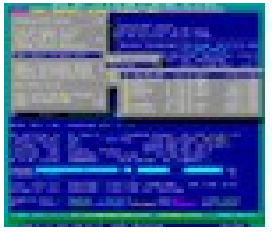
  - If all that fails, contact me at:
    - info@dfsee.com

TxWin 5.xx

Programming and User Guide

Questions ?

FSYS - Software



TxWin