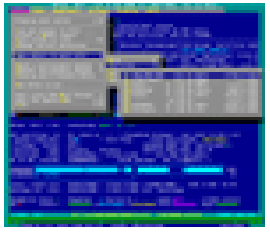
Open TxWindows Installation and Samples

Jan van Wijk

Overview of installation, configuration and sample programs that come with the open-source TxWindows library

FSYS - Software



TxWin

Presentation contents

- What & Why, Open TxWindows library
- Distribution and packaging
- Compiler installation and setup
- A standard build environment for programs
- Samples from the straight-C “Hello world” upto a windowed text-viewer with menus and file dialogs ...

What & Why, Txwindows

- TxWindows is a library to be used from 'C' or C++, implementing text-mode windowing for several operating system platforms.
  - Includes several non windowing related modules like parameter parsing, tracing, directory and file itterators, command interpreters and scripting.
- Development started almost 10 years ago, for my LPTool and DFSee programs.
  The effort was needed because easy to use, portable and powerful text-mode libraries could not be found ...

Distribution and packaging

- Ready to use versions of the library will be made available as ZIP-files, including all required files to develop and build and test programs using the library (build on OS/2 only for the moment)
- Until release of OpenWatcom 1.4, an OS/2 build of that compiler will be provided too.
  - OW 1.3 works too, but cannot create Linux targets
- A CDROM-ISO and/or Warpin installer archive may be made available too (time permitting :-)
- Eventually, the sources will be available and maintainable through NetLabs ...

Compiler install and setup

- The compiler uses the standard OpenWatcom directories, and is simply copied / unzipped.
- NO updates to config.sys are required. Instead all needed settings are defined in a script that results in a specific OW / TxWin command window
  - TxWindows works with the VisualAge C++ compiler as well, but has not been built for that environment for a while since that compiler is obsolete, and not cross-platform ...
  - It should not be hard to port it for use with the GCC compiler, available for all supported platforms, it is NOT a cross-compiler though, so you need to build on every platform seperately ...

Standard build environment

- Allows quick creation of new projects by deriving them from previous ones (or from samples)
- Includes the master makefile (MIF), set up for cross-compilation to all required targets with minimal changes required between projects
- Building further automated using a few standard scripts, (nearly) identical for every program.
- Purely commandline based, no IDE :-)

Multi platform / trace variants

- The environment currently supports:
  - OS/2 (or eCS), as 32-bit executables (OS2 2.0 and up)
  - Windows 32-bit console mode (Win-NT and later)
  - DOS, using 32-bit dos-extended executables
  - Linux 32-bit executable, runs in console and XTERM
- For each of these OS platforms, you get:
  - A 'retail' version, no trace or application trace only
  - A 'trace' version, with trace in application and library
  - A 'debug' version, compiled for use with the debugger
- Meaning you end up with 12 executables ...

New project, HOWTO derive

- Creating a new project from a similar one:
  - Recursive copy of directory tree (xcopy /s)
  - Rename the main sourcefile (project.c)
  - Update master makefile.mif (compo=...)
  - Remove old binaries: 'b all clean'
  - Build new default target: 'b'
  - Make other functional changes, starting the actual development cycle

The development cycle

- Start of cycle, phase 1
  - Make changes to the source(s) or makefile(s)
  - Build default target 'b'
  - If any compile/link errors, analyse, fix and retry
  - Test / trace / debug target when built OK
  - If changes required, back to start of cycle ...
- Cycle, phase 2
  - Build all targets ' b all'
  - If any compile/link errors, analyse, fix and retry
  - When OK, test these targets on each platform

Available samples

- Included with the TxWindows distributions are some sample projects ranging from the trivial hello-world to an almost usable text viewer application :-)
- They all share the same directory structure and build-mechanism, and are good candi-dates for deriving other projects
- The samples are also good for practising trace and debug in this environment

TXT, the TxWindows test program

- Finally, there is a larger program primarily made for testing TxWindows itself
- It is useful for testing additions and changes to the library, and should be extended for significant new functionality
- Also very useful to check look-and-feel, behaviour and trace/debug capabilities.

SAM1, Hello World

- This is the classic 'C' program used to verify your compiler and build environment.
- It does NOT use the TxLibrary at all
- Test and demo: Use of the WD debugger

Sam2, Hello Trace

- Functionally the same program, but using the TxWindows library to add:
  - Tracing, and standard 'main' processing
  - Standard argument handling
  - Standard logging to ASCII file
  - Usage help with th '-?' switch
- Test and demo: Trace to screen / file

Sam3, Hello Window

- Extends the previous sample with:
  - Hello message in a Window with [OK] button
- Demo and test, tracing:
  - sample -123sd test-see-popup
    - Screen trace OFF when popup starts ...
  - sample -345sd55 -p test-scroll-off
    - Screen trace remains ON during popup ...

Sam4, Hello Scrollbuffer

- Extends the previous sample with:
  - A large scroll-buffer for all regular output, which is just the trace-output in this sample.
  - Demonstrates using that, and shows switching from STDOUT to the windowed environment on-the-fly while tracing ...
- Test and demo:
  - Show effect of invalidate() (on scrollbuffer)
    - Requires source update and build ...

Sam4, more test and trace

- Some more interesting tests:
  - 'sample -222sd -p test one two'
    - Then, when popup is there, use <F12>
  - 'sample -0d -p test' (that is a zero :-)
    - '<Alt>+/' to toggle trace to title-area and buffer
      - No trace => quick reaction, scrolling
      - Title-trace => slow tracing to title-line
      - Screen trace => to scroll-buffer
    - '<Al>+m + Arrow-keys to demo move processing

Sam5, Hello Text Viewer

- This implements a very basic text viewer with text from a specified file shown in a standard TxWindows text-view class
- No scroll-buffer, view window directly on top of the (transparent) desktop window.
- Inludes tracing and argument handling

Sam6, Hello File Dialog

- This extends the previous sample with a standard File-Open dialog
  - Dialog when no parameters given or <Ctrl>+O
  - Usage help with explicit '-?' argument
  - Filename and number of lines shown in title, loading a new file replaces the current file
  - Uses a window-procedure to implement handling of the special keys and file dialog

Sam7, Hello Menu and Help

- This is the most complete sample, adding:
  - A popup menu-bar with a main menu
  - Help screens for menu items and viewer window
  - An 'About ...' popup window in the help menu
  - Text artwork as initial viewer background
  - Explicit key-handling in window-procedure replaced by accelerator definitions, sharing processing with the corresponding menu items

Trace, HOWTO use

- Functional tracing is built into TxWindows, and hopefully the application too. A VERY powerful mechanism for trouble shooting!
- To be refined :-)

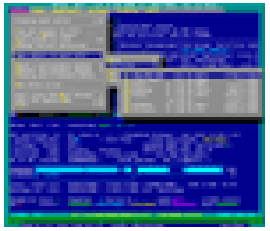
Debugger HOWTO use

- In the rare situations that TRACE does not work well to find a problem, use WD ...
- To be refined :-)

Open TxWindows
Installation and Samples

Questions ?

FSYS - Software



TxWin