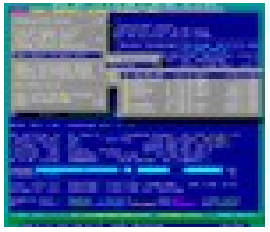
TxWin 5.xx

Installation and Samples

Jan van Wijk

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

FSYS - Software



TxWin

Presentation contents

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

What & Why, TxWindows

- **TxWin** is a free 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 iterators, command interpreters and scripting.
- Development started over 25 years ago, for my LPTool and DFSee programs I was developing.

  Needed because portable, easy to use and still powerful text mode libraries could not be found ...

Distribution and packaging

- Ready to use libraries are available as ZIP-files, including all required files to develop, build and test programs (built on macOS with OS/2 in a VM)
- Tested with OpenWatcom 1.9 and 2.0 releases, on both OS/2 (ArcaOS) and Win-XP/Win-7
  - Releases from 1.4 onward should work (Linux support).
- The macOS version of the library and samples have been built on macOS 10.13 (High Sierra) with the Xcode and CLANG versions for that.
  - Older versions have been built with previous versions without any problem too, back to when gcc/gdb where still current
  - May work with GCC 4.2 or newer on Linux too (untested)

Compiler install and setup

- The compiler uses the standard OpenWatcom directories, and is simply copied / unzipped.
- All needed settings are defined in a script that results in a specific OW / TxWin CMD window
  - Included script is for OS/2, Windows is very similar
  - No special setup is needed for use with the GCC compiler, available for all supported platforms, however only GCC on macOS using regular makefiles has ever been tested
  - The latest macOS development environments have switched to the CLANG/LLVM tools suite, compatible with GCC

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 scripts
- OpenWatcom command line based, no IDE :-)
- Using 'make' and makefile.osx master makefile on macOS, and Xcode GUI for build/debug

Multi platform build variants

- The environment currently supports:
  - OS/2 (or ArcaOS/eCS), 32-bit exe (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
  - MacOS 64 bit executables, runs in Terminal or iTerm
- For each of these OS platforms, you can get:
  - A 'retail' version, but including full functional trace
  - A 'debug' version, compiled for use with the debugger
- Meaning you may end up with 10 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=...)
    - (similar changes to makefile.osx)
  - Remove old binaries: 'b all clean'
  - Build new default target: 'b'
    - macOS: 'make all clean'
      'make'
  - 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'   or macOS: 'make'
  - If any compile/link errors, analyze, 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' or macOS: 'make all'
  - If any compile/link errors, analyze, 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-mechanisms, and are pretty good candidates for deriving other projects
- The samples are also good for practising trace and debug in this environment

TXT, the TxWin test program

- Finally, there is a larger program primarily made for testing TxWin 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 TxWin library at all
- Test and demo: Use of the WD debugger
  (Or the Xcode/LLDB debugging GUI on macOS)

Sam2: Hello Trace

- Functionally the same program, but using the TxWin library to add:
  - Tracing, and standard 'main' processing
  - Standard argument handling
  - Standard logging to ASCII file
  - Usage help with the '-?' 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 Output Window

- 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 scroll buffer)
    - 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
    - '<Alt>+<F7> + 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 output window, view window directly on top of the (transparent) desktop window.
- Includes 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

Sam8: Output , Menu and Help

- This is another complete sample, with:
  - A popup menu-bar with a main menu
  - Help screens for menu items and viewer window
  - An 'About ...' popup window in the help menu
  - An output window (scroll buffer) for any output from commands to be executed
  - A command line to enter commands

Sam9: Script expression evaluator

- This is small sample to test expressions
  - Type any expression as used with TX-script
  - The expression is evaluated and the result shown
  - Includes support for variables and constants

Trace, HOWTO use

- Functional tracing is built into TxWin, and hopefully the application too. It is a VERY powerful mechanism for trouble shooting!
- Trace is started when starting the application by using a '-nnn' switch where nnn is the trace-level, or a 'trace' command with the level and/or filename as parameters
  (Use 'trace -?' for help on the trace command)
- To be refined :-)

Debugger HOWTO use

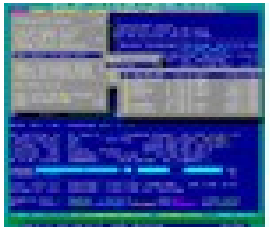
- In the rare situations that TRACE does not work well to find a problem, use WD as a text mode debugger on OS/2 or the Xcode GUI with LLDB on macOS.
  - The Xcode GUI being more modern and powerful
- To be refined :-)

TxWin 5.xx

Installation and Samples

Questions ?

FSYS - Software



TxWin