Building Stroika

Common

Stroika is a C++ class library. For the most part, it's built make. However, internally, some of these make rules use perl etc scripts.

Quick Start

For the very impatient

```
wget https://github.com/SophistSolutions/Stroika/archive/V2-Release.tar.gz
tar xf V2-Release.tar.gz
make --directory Stroika-2-Release all run-tests
```

For the more patient (hints about what to try next)

- See Required Tools below (or wait for warnings during the build process)
- wget https://github.com/SophistSolutions/Stroika/archive/V2-Release.tar.gz
  or
  wget https://github.com/SophistSolutions/Stroika/archive/v2.0a211.tar.gz
- tar xf V2-Release.tar.gz
- cd Stroika-2.0a211 (or whatever version extracted)
- make help
  Not needed, but gives some idea of make options
- make check-tools
  Not needed, but tells you if you are missing anything critical
- make default-configurations
  Not needed, but it's a springboard for setting up the configuration you want.
  - Review/edit ConfigurationFiles/Debug.xml
  - Or try something like
    - configure MyGCC63Config -assertions enable --trace2file enable --compiler-driver 'g++-6.3'

    or
    - configure MyCPP17Test --assertions enable --trace2file enable *--cppstd-version-flag '--std=c++17'*

    or
    - configure --help
- ls ConfigurationFiles/
  - See what configurations you've created. Edit files, add or delete.
- make all CONFIGURATION=*a-config*
  - make all targets for the configuration named *a-config*
- make all
  Builds everything for ALL configurations (in the folder ConfigurationFiles/)
- make run-tests
  Runs regression tests (optionally on remote machines, or with VALGRIND)

Required Tools

Required for ALL platforms

- c++ compiler supporting C++14 or later
- make (gnu make)
- patch
- perl
- pkg-config
- realpath
- sed
- tar
- tr
- wget
- 7za (if building with LZMA SDK – the default)
- unzip

For MacOS

- XCode 8, XCode 9 or later
    - install from appstore,
    - Then from command line
        - xcode-select –install
        - Homebrew can be helpful (but use whatever package mgr you wish)
            - ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
            - to install apps with brew, use “brew install APPNAME”
        - gnu-sed
        - p7zip (if building lzma)

For UNIX

- Compiler
    - gcc 6 or later OR
        - Stroika is currently tested with gcc 6.x, 7.x
    - llvm (clang++) 3.9 or later
- automake (if building curl)
- libtool (gnu version) – (if building curl)

For Windows

- Visual Studio.net 2017 (or later)
- Cygwin
  Including
  - dos2unix
  - unix2dos

Third Party Components

These components automatically downloaded and built and integrated into Stroika (depending on configuration) or Stroika can be configured to use the system installed version of these components.

- curl
- openssl
- zlib
- lzma SDK
- xerces

Build / Configuration Design

Alternatives

We seriously considered a number of build systems, including cmake, ant, perl scripts, qmake, etc. They all weak, with ant possibly being the best alternative, except that it's heavily java oriented.

Just normal GNU make – appears to be the least bad alternative, so that's what we're doing.

Configurations

The design is to make a set of configuration files – each stored in the ConfigurationFiles/ directory. Configurations are described by a single XML file. They can be generated using the configure script.

```
configure --help
```

for more information.

A configuration comprises BOTH target platform (e.g. windows/linux), hardware (e.g. PowerPC, x86, x64, ARM), and any other options (debug build, enable assertions, support OpenSSL, etc).

This stands in contrast to several other systems (like Visual Studio.net) that treats platform and 'configuration' as orthogonal choices.

Configuration files can be edited by hand.

@todo A FUTURE version of Stroika will have an XML Schema (XSD) to describe the configuration file format.

You can call

```
make apply-configurations
```

to generate all the directories and files dependent on the defined configurations. Note – this is generally not necessary, and called automatically.

Build Results

Intermediate files (objs etc) go into **IntermediateFiles/{CONFIGURATION-NAME}**. Final build products (libraries and executables) go into **Builds/{CONFIGURATION-NAME}**.

Build Process

On any platform, building Stroika, and all is demo applications and regression tests is as simple as cd'ing to the top-level directory, and typing make

Special Targets

- make
  Make with no arguments runs 'make help'
- make help
  Prints the names and details of the special targets
- make all
  builds the stroika library, tests, demos, etc.
- make libraries
  Builds just the Stroika libraries
- make samples
  Builds the Stroika sample applications
- make run-tests
  Builds Stroika, and all the regression tests, and runs the regression tests
- make project-files
  Builds project files which can be used for things like visual studio (not needed)
- make check-tools
  Checks if the tools needed to build Stroika are installed and in your path. This is done automatically, and generally not needed explicitly.

Configuration arguments to make

All the make targets (e.g. all, libraries etc) take an OPTIONAL parameter CONFIGURATION. If specified, only that configuration is built. If omitted (or empty) – ALL configurations are built.

Using Visual Studio.net

Visual Studio.net project and solution files are available for the Stroika demos, top-level project files, and regression tests. Once you have built your configuration files (see above), you can use the project files to build, test, extend and develop Stroika.

Using QtCreator (on unix)

Run Library/Projects/QtCreator/CreateQtCreatorSymbolicLinks.sh to create project files at the top level of your Stroika directory. Then you can open that .creator file in qtCreator, and build and debug Stroika-based applications.

Building For…

Building for Raspberry Pi

To cross-compile for Raspberry pi,

- install some suitable cross compiler (in this example `arm-linux-gnueabihf-g++-5`)

  On unubtu, sudo apt-get install g++-5-arm-linux-gnueabihf

- `configure raspberrypi-gcc-5 --apply-default-debug-flags --trace2file enable --compiler-driver arm-linux-gnueabihf-g++-5 --cross-compiling true`

  Set cross-compiling true so that internal tests aren't run using the arm built executables.
  Set –apply-default-release-flags instead of 'debug' for a smaller faster executable.
  --trace2file disable to disable tracefile utility, and enabled writes a debug log to /tmp.

- `make CONFIGURATION=raspberrypi-gcc-5 all`

  This builds the samples, libraries etc to Builds/raspberrypi-gcc-4.9)

- `make run-tests REMOTE=pi@myRasberryPi`

  This uses ssh to run the tests remotely on the argument machine (I setup a hostname in /etc/hosts for the mapping).

  Using SSH, it's also helpful to setup ssh keys to avoid re-entering passwords

  http://sshkeychain.sourceforge.net/mirrors/SSH-with-Keys-HOWTO/SSH-with-Keys-HOWTO-4.html

Common Errors

Tar failure

Errors about invalid parameters, and/or bad blocks can usually be fixed by installing a copy of gnu tar. We've tested 1.27.

cp: illegal option –

Install a copy of GNU cp

Cannot find 'blah' in Cygwin

If you are trying to install required components in Cygwin, and cannot find them in the Cygwin setup GUI, try:

```
cygcheck -p dos2unix
```