STLSoft — Getting Started Guide

revision 79, 15 Jul 2008
STLSoft version 1.9.44

Matthew Wilson

Martin Moene

Contents

STLSoft Official Website: http://stlsoft.org/

Preface

Mini Contents

Audience

This guide is meant for software developers and managers who are considering to employ the STLSoft C++ libraries in their project. The guide intends to be useful to decide if a project may benefit from using STLSoft and to help to get the gist of programming with STLSoft.

For the first three chapters —**Introduction**, **Library Overview**, and **Extending STL**— general knowledge of software engineering concepts and a basic understanding of the C++ Standard Template Library is all is needed. To appreciate the later chapters about using STLSoft, it's beneficial to also have some experience with C++ and templates.

Acknowledgments

The following free software has been used to create this guide:

- Code::Blocks IDE
- Doxygen documentation system
- Firefox Web browser
- GNU C C++ compiler
- Subversion version control system
- Thunderbird e-mail and news client
- TortoiseSVN Subversion windows shell extension

Introduction

Mini Contents

What is STLSoft?

STLSoft is a collection of open source C++ template libraries that extend the functionality of the Standard Template Library and provide facades that wrap operating-system and technology-specific APIs.

The components that STLSoft provides are meant to be used as *building blocks* for higher level components such as applications, classes, libraries and servers. STLSoft is not a framework .

STLSoft is organized along two lines: projects and libraries. Projects generally relate to platforms, whereas libraries relate to technology areas. Examples of projects are ACESTL, UNIXSTL, WinSTL, PlatformSTL. Examples of libraries are the Performance Library, the Smart Pointers Library and the Windows Registry Library. Projects and libraries are explained in more detail in chapter **Library Overview**.

STLSoft covers various Windows and Unix/Linux operating systems and works with a wide variety of compilers. STLSoft is written in C++, but it also contains parts that can be used with C. The libraries use templates and rely on compile-time polymorphism. The STLSoft libraries are header-only: the library code is included in and compiled with the user's code. This makes **installation** and usage of the libraries relatively easy.

STLSoft is licensed under a BSD-form **license** to allow it to be used with both open and closed source projects.

What are STLSoft's origins?

Early in 2002, STLSoft started as an open source project. Many parts of the libraries were derived from proprietary Synesis Software code.

Why should I use STLSoft?

There may be many reasons to use STLSoft. Several of them are that STLSoft

- contains a wealth of useful components
- enables and supports working the STL-way
- covers mutiple platforms
- is partially cross-platform
- works with many compilers
- is highly portable
- is lightweight
- is fast
- has high cohesion
- has minimal coupling
- has been extensively used in open and closed source projects
- uses a BSD-form **license**

The STLSoft libraries cover operating systems diverse as Microsoft Windows, Linux/Unix, Mac OS-X and Sun Solaris several of which both in 32-bit and 64-bit versions.

Furthermore, STLSoft works with many compilers: Borland C++, CodePlay VectorC C/C++, Comeau C/C++, Digital Mars C/C++, GNU C, Intel C/C++, Metrowerks C/C++, SunPro C/C++, Visual C++ and Watcom C/C++.

See also appendix **B. Supported Platforms**

A tiny example

The power of STL abstractions offered by STLSoft is nicely illustrated by the following example. The program reads and displays the directories present in the root directory whether the program is compiled on Unix or compiled on Windows. It uses the class `readdir_sequence` from STLSoft's cross-platform project PlatformSTL that adapts the Unix `opendir/readdir` API and the `FindFirstFile/FindNextFile` Windows API to an iteratable sequence.

```
#include <platformstl/filesystem/readdir_sequence.hpp>

#include <algorithm>    // std::copy
#include <iterator>     // std::ostream_iterator
#include <iostream>     // std::cout, std::endl
```

```
#include <string>       // std::string

using platformstl::readdir_sequence;

int main()
{
   readdir_sequence dir( "/", readdir_sequence::directories );

   std::copy
   ( dir.begin()
   , dir.end()
   , std::ostream_iterator< std::string >( std::cout, "\n" )
   );
}
```

Using the GNU C compiler, the executable program can be build as follows with both Unix and Windows:

```
g++ -Wall -I../stlsoft-1.9.44/include program.cpp -o program
```

Here it is assumed that the STLSoft library (release 1.9.44) is installed in a sibling directory of the directory that contains the program.

As a stark contrast to the example above, you may want to see a native Unix implementation to make such a directory listing.

Where to get STLSoft

The STLSoft libraries can be downloaded from the STLSoft website: http://stlsoft.org/ . Besides the latest release and older releases, the website also contains the latest reference documentation of the libraries.

How to install STLSoft

Since STLSoft is implemented as a collection of header-only libraries, no real installation procedure is required. It suffices to unzip the archive stlsoft-1.9.44-hdrs.zip in a suitable location, for example in /home/*myaccount*/libraries/ on Unix or in D:\Libraries\ on Windows. The archive unpacks in a subdirectory named `stlsoft-1.9.44`, so that it is possible to use different versions of the STLSoft libraries side-by-side.

Next, define the environment variable STLSOFT to point to the STLSoft toplevel directory. For example, enter one of the the following commands in the shell or the command box:

```
set    STLSOFT=D:/Libraries/stlsoft-1.9.44   & rem for Windows cmd
export STLSOFT=D:/Libraries/stlsoft-1.9.44   # for Bourne, bash, and related shells
setenv STLSOFT D:/Libraries/stlsoft-1.9.44   # for csh and related shells
```

Note that Windows also *can* use forward slashes in the path.

The commands above creates a temporary environment variable. See the article Environment variable on Wikipedia to learn how to make a permanent entry for the evironment variable on your operating system.

Using the environment variable just created, the command to compile a program may look as follows.

On Unix:

```
g++ -Wall -I"$(STLSOFT)/include" program.cpp -o program
```

On Windows:

```
g++ -Wall -I"%STLSOFT%/include" program.cpp -o program
```

Now let's turn to the **next chapter** to see what STLSoft has to offer.

Library Overview

Mini Contents

Before we take a closer look at some of SLSoft's components in following chapters, let us first form an idea about the kind of things STLSoft contains and how to find our way in the sea of components.

Projects and Libraries

In the **first chapter** we already learned that STLSoft is organized in two ways:

1. as **projects**, for example: COMSTL, PlatformSTL, UNIXSTL and WinSTL
2. as **libraries**, for example: File System Library, Performance Library, Synchronisation[1] Library

There are 12 projects and 25 libraries. Appendix **A. Library Contents** lists all projects and libraries.

Projects The main project is also called STLSoft and it contains most platform- and technology-independent code and code that helps to take care of compiler and standard library differences.

The biggest projects are COMSTL, UNIXSTL and WinSTL. COMSTL provides utilities to work with the Component Object Model (COM) and provides STL-compatible sequence adaptors over the COM enumeration and COM collection concepts. UNIXSTL and WinSTL provide operating-system and technology-specific components for Unix and Windows operating systems. These two projects have a number of structural conformant components, such as `environment_variable`, `path` and `thread_mutex` that are also placed in the PlatformSTL project that facilitates writing platform-agnostic code.

The other projects address additional technology-specific areas. ACESTL applies STL concepts to some components of the Adaptive Communication Environment (ACE) library. MFCSTL makes using the Microsoft Foundation Classes (MFC) more STL-like. RangeLib is the STLSoft implementation of the range concept. InetSTL, ATLSTL and WTLSTL are smaller projects that apply the STL concepts to Internet programming and to programming with the Active Template Library (ATL) and Windows Template Library (WTL) .

Libraries The subjects covered by the STLSoft libraries are quite diverse. The next section will gently introduce you to this variety of subjects.

The STLSoft Matrix

When you are new to STLSoft, the number of components that comprise STLSoft may seem overwhelming. To get familiar with it and gain more insight into the relations that exist within STLSoft, it may help to view part of its contents as a matrix. In this matrix, the columns show a project-oriented context and the rows show a library-oriented context.

The table below lists all libraries, but it doesn't show projects that only cover specialistic areas such as ACESTL and WTLSTL.

In the table, the symbol at each cross-section of the matrix indicates if a library has components in the project (+), if a library is only relevant for one project (1), or if several projects have structurally conformant components in the library (c).

Library — Project	STLSoft	COMSTL	UNIXSTL	WinSTL	PlatformSTL
Collections Library	+	+			
Containers Library	+				
Conversion Library	+	+		+	
DL Library			—c—	—c—	—c—
Error Library	+	+	—c—	—c—	—c—
File System Library	+		—c—	—c—	—c—
Functional Library	+	+		+	
Iterators Library	+				
Memory Library	+	+		+	
Template Meta-programming Library	+				
Performance Library	+		—c—	—c—	—c—
Properties Library	+				
Security Library		+		+	
Smart Pointers library	+	+			
String Library	+	+		+	
Synchronisation Library	+		—c—	—c—	—c—
System Library	+		—c—	—c—	—c—
Time Library				+	
Utility Library	+	+		+	
Windows Clipboard Library				1	
Windows Control Panel Library				1	
Windows Controls Library				1	
Windows Registry Library				1	
Windows Shell Library				1	
Windows ToolHelp Library				1	
Library — Project	STLSoft	COMSTL	UNIXSTL	WinSTL	PlatformSTL

+ has components c structural conformant 1 single platform

Consider the matrix as a starting point for your explorations of the STLSoft libraries. The combination of platform –project– and library name –functionality– may help to direct your attention to certain parts of STLSoft. Then continue your journey with the corresponding **examples** and reference documentation.

Don't leave yet, in a moment we'll be developing a small program that uses STLSoft together. But first we'll see how STLSoft **extends the Standard Template Library** and look at several **short STLSoft examples**.

---

^[1] British-English spelling is used as in initialiser and tokeniser.

Prev Introduction | **Next** Extending STL

Extending STL

Mini Contents

STLSoft extends the Standard Template Library (STL) , so before we delve into how STLSoft extends STL, let's recap STL's concepts.

Standard Template Library

Core concepts The STL is based on six core concepts: containers, iterators, algorithms, functions objects, adaptors and allocators.

Containers Containers store objects. There are four sequence containers, three sequence container adaptors, and four associative conainers. The sequence container are deque, `list`, `vector` and `basic_string`. These containers store their elements in a linear arrangement. The sequence container adaptor class templates are queue, `priority_queue`, and `stack`. These are *class adaptor* types. The associative containers are `map`, `multimap`, `set`, and `multiset`. These containers provide lookup of elements based on a key. (Four unordered variants, that use hash table lookup, will be added in C++0x .)

Iterators An iterator is an object that is used to traverse through the elements of a container, regardless of its specific implementation. There are five iterator categories: input iterator, output iterator, forward iterator, bidirectional iterator, and random access iterator. Iterators are the link between containers and the algorithms working on their data. There are also iterator adaptors, for example `std::reverse_iterator`, which is a *class adaptor*, and `std::back_insert_iterator` and `std:istream_iterator`, which are iterator *instance adaptors*.

Algorithms STL algorithms are generic function templates that are applied to iterator ranges and operate on the elements within them or on the range itself. Examples of algorithms are `std::transform` and `std::distance`.

Functions Objects A function object , or functor is a function or an object of a class that implements the function call operator. Funtion objects are primarily used in combination with algorithms. They occur as predicates , for example with algorithm `count_if`, or as function, for example when used with `std::transform`.

Allocators Allocator objects are used for memory management for containers.

STLSoft

Collections

Shims and Veneers

Shims

Veneers

Many ideas and techniques that STLSoft uses are described in [**Imperfect C++**] and in [**Extended STL, Volume 1**].

Prev Library Overview | **Next** STLSoft Examples

STLSoft Examples

This chapter contains a selection of short example programs for the wide range of components that STLSoft offers. It is a long chapter, but just skimming it already may give a good idea of STLSoft's contents. Appendix **C. Compiling Examples** presents the makefiles used to compile these examples.

Mini Contents

ACESTL Project

Sorry, no examples yet.

ATLSTL Project

Sorry, no examples yet.

COMSTL Project

COMSTL Library (collection sequence)

COMSTL Library (enumerator sequence)

InetSTL Project

InetSTL Library (findfile sequence) Note there also is a more recent class `inetstl::ftpdir_sequence` that is similar to this class, but has some optimisation built-in.

```
#define VC_EXTRALEAN    // exclude rarely-used stuff from Windows headers

#include <afxwin.h>     // MFC core and standard components
#include <wininet.h>    // Internet functions

#include <inetstl/filesystem/findfile_sequence.hpp>
#include <stlsoft/smartptr/scoped_handle.hpp>
#include <stlsoft/string/static_string.hpp>

#include <algorithm>    // std::for_each
#include <stdexcept>    // std::logic_error
#include <iostream>     // std::cout, std::endl

typedef stlsoft::scoped_handle< HINTERNET > scoped_internet_handle;

HINTERNET open_internet();
HINTERNET connect_ftp( HINTERNET hConnection, const char* hostname );

void print_name( inetstl::findfile_sequence::value_type const& v )
{
   std::cout << "name: " << v.get_short_filename() << std::endl;
}

int main()
{
   using inetstl::findfile_sequence;

   try
   {
      const char* hostname = "ftp.eld.leidenuniv.nl";

      scoped_internet_handle  ip_connection( open_internet(), ::InternetCloseHandle );
      scoped_internet_handle ftp_connection( connect_ftp( ip_connection.get(), hostname ), ::InternetCloseHandle );

      findfile_sequence files( ftp_connection.get(), "/", "*.*", findfile_sequence.files );

      std::for_each( files.begin(), files.end(), print_name );
   }
   catch ( inetstl::internet_exception const& e )
   {
      std::cerr << "InetSTL Internet error: " << e.what() << ", code " << e.get_error_code() << std::endl;
   }
   catch ( std::exception const& e )
   {
      std::cerr << "Error: " << e.what() << std::endl;
   }
   catch ( ... )
   {
      std::cerr << "Unknown exception" << std::endl;
   }
}

// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - =

HINTERNET handle_error( HINTERNET handle, std::string message )
{
   if ( NULL != handle )
```

```
   {
      return handle;
   }

   DWORD error;
   stlsoft::basic_static_string< TCHAR, _MAX_PATH > path( "", _MAX_PATH );
   DWORD length = path.size();

   BOOL status = ::InternetGetLastResponseInfo
   ( &error
   , &path[0]
   , &length
   );
   path.resize( length );

   throw std::logic_error( message + ": " + ( path.size() ? path.c_str() : "(no details)" ) );
}

HINTERNET open_internet()
{
   HINTERNET handle = ::InternetOpen
   ( "STLSoft"                  // name of application
   , INTERNET_OPEN_TYPE_PRECONFIG//
   , NULL                       // read proxy information from the registry
   , NULL                       // read the bypass list from the registry.
   , 0                          // no flags
   );

   (void) handle_error( handle, "cannot open internet connection" );

   DWORD flags;
   if ( ! ::InternetGetConnectedState( &flags, 0 ) )
   {
      throw std::logic_error( "no i-net connection available" );
   }

   return handle;
}

HINTERNET connect_ftp( HINTERNET hConnection, const char* hostname )
{
   HINTERNET handle = ::InternetConnect
   ( hConnection                // internet session
   , hostname                   // server name
   , INTERNET_DEFAULT_FTP_PORT  // server port
   , NULL                       // username: anonymous
   , NULL                       // password: e-mail address
   , INTERNET_SERVICE_FTP       // dwService
   , INTERNET_SERVICE_FTP       // dwFlags
   , 0                          // dwContext
   );

   return handle_error( handle, "cannot open ftp connection" );
}

// cl -MDd -W3 -EHsc -D"_AFXDLL" -I"%STLSOFT%/include" inetstl_findfile_sequence.cpp wininet.lib
```

MFCSTL Project

MFCSTL (CArray class adaptor) Adapted from STLSoft documentation.

```
#define VC_EXTRALEAN    // exclude rarely-used stuff from Windows headers

#include <afxwin.h>     // MFC core and standard components

#include <mfcstl/collections/carray_adaptors.hpp>

#include <algorithm>    // std::copy
#include <iterator>     // std::ostream_iterator
#include <iostream>     // std::cout, std::endl
#include <list>         // std::list

int main()
{
   mfcstl::CArray_cadaptor< CStringArray > array;

   // as an MFC CStringArray:
   array.Add     (    "String 1" );
   array.InsertAt( 0, "String 0" );

   // as an STL container
   array.push_back( "String 2" );

   std::list< CString > slist;
```

```
   slist.push_back( "String 4" );
   slist.push_back( "String 3" );

   array.insert( array.begin() + 2, slist.begin(), slist.end() );

   std::sort( array.begin(), array.end() );

   std::copy
   ( array.begin()
   , array.end()
   , std::ostream_iterator< const char* >( std::cout, "\n" )
   );
}

// cl -MDd -W3 -EHsc -D"_AFXDLL" -I"%STLSOFT%/include" mfcstl_carray_cadaptor.cpp
```

MFCSTL (CArray instance adaptor) Adapted from STLSoft documentation.

```
#define VC_EXTRALEAN    // exclude rarely-used stuff from Windows headers

#include <afxwin.h>     // MFC core and standard components

#include <mfcstl/collections/carray_adaptors.hpp>

#include <algorithm>    // std::copy
#include <iterator>     // std::ostream_iterator
#include <iostream>     // std::cout, std::endl
#include <list>         // std::list

int main()
{
   CStringArray                            array;
   mfcstl::CArray_iadaptor< CStringArray > ardap( array );

   // as an MFC CStringArray:
   array.Add     (    "String 1" );
   array.InsertAt( 0, "String 0" );

   // as an STL container:
   ardap.push_back( "String 2" );

   std::list< CString > slist;
   slist.push_back( "String 4" );
   slist.push_back( "String 3" );

   ardap.insert( ardap.begin() + 2, slist.begin(), slist.end());

   std::sort( ardap.begin(), ardap.end() );

   std::copy
   ( ardap.begin()
   , ardap.end()
   , std::ostream_iterator< const char* >( std::cout, "\n" )
   );
}

// cl -MDd -W3 -EHsc -D"_AFXDLL" -I"%STLSOFT%/include" mfcstl_carray_iadaptor.cpp
```

PlatformSTL Project

PlatformSTL DL Library (module) See also **WinSTL DL Library (dl call)**.

```
#include <platformstl/dl/module.hpp>

#include <iostream>     // std::cout, std::endl
#include <string>       // std::string

using platformstl::module;

std::string get_system_dir()
{
   typedef DWORD (WINAPI *function_t)( LPTSTR, UINT );

   module mod( "KERNEL32.DLL" );

   function_t function;
   mod.get_symbol( "GetSystemDirectoryA", function );

// or:
//   function_t function = (function_t) mod.get_symbol( "GetSystemDirectoryA" );

   if ( NULL != function )
```

```
   {
      TCHAR dir[ 1 + _MAX_PATH ];

      DWORD length = function
      ( &dir[0]                     // parameter 1
      , STLSOFT_NUM_ELEMENTS( dir ) // parameter 2
      );
      return 0 == length ? "[empty]" : dir;
   }
   return "[no such function]";
}

int main()
{
   std::cout << "System directory: " << get_system_dir() << std::endl;
}
```

PlatformSTL Filesystem Library (path)

```
#include <platformstl/filesystem/path.hpp>

#include <iostream>     // std::cout, std::endl

using platformstl::path;

#define DO( cmd, result ) \
   std::cout << #cmd << ": " << cmd << " --- " << result << std::endl;

int main()
{
   DO( path( "platformstl_path.cpp" ).exists()   , "(yes)"      );

   DO( path(   "path/to/file.ext").get_file()    , "(file.ext)" );
   DO( path(   "path/to/file.ext").get_ext()     , "(ext)"      );

   DO( path (  "path/to/file.ext").is_rooted()   , "(no)"       );
   DO( path(  "/path/to/file.ext").is_rooted()   , "(Unix: yes, Windows: yes)" );
   DO( path("D:/path/to/file.ext").is_rooted()   , "(Unix: ---, Windows: yes)" );

   DO( path(   "path/to/file.ext").is_absolute() ,  "(no)"      );
   DO( path(  "/path/to/file.ext").is_absolute() ,  "(Unix: yes, Windows: no)" );
   DO( path("D:/path/to/file.ext").is_absolute() ,  "(Unix: ---, Windows: yes)" );

   DO( path( "/path/to/sub/../file.exe" ).equivalent(       "/path/to/file.exe"  ), "(yes)" );
   DO( path( "/path/to/sub/../file.exe" ).canonicalise() , "(/path/to/file.ext)" );
   DO( path( "platformstl_path.cpp"     ).make_absolute(), "(Unix: /..., Windows: D:\\....)" );
}
```

PlatformSTL Filesystem Library (pipe)

```

```

PlatformSTL File System Library (readdir sequence)

```
#include <platformstl/filesystem/readdir_sequence.hpp>

#include <algorithm>    // std::copy
#include <iterator>     // std::ostream_iterator
#include <iostream>     // std::cout, std::endl
#include <string>       // std::string

using platformstl::readdir_sequence;

int main()
{
   readdir_sequence dir( "/", readdir_sequence::directories );

   std::copy
   ( dir.begin()
   , dir.end()
   , std::ostream_iterator< std::string >( std::cout, "\n" )
   );
}
```

PlatformSTL Performance Library (performance counter)

```
#include <platformstl/performance/performance_counter.hpp>
```

```
#include <iostream>     // std::cout, std::endl

int main()
{
   platformstl::performance_counter counter;

   counter.start();
   for ( volatile size_t i = 0; i != 0x7fffffff; ++i )
      ;
   counter.stop();

   std::cout << "Number of seconds:      " << counter.get_seconds() << std::endl;
   std::cout << "Number of milliseconds: " << counter.get_milliseconds() << std::endl;
   std::cout << "Number of microseconds: " << counter.get_microseconds() << std::endl;
}
```

PlatformSTL Synchronisation Library (critical section)

```
#define VC_EXTRALEAN    // exclude rarely-used stuff from Windows headers

#include <windows.h>    // for DWORD HANDLE

#include <platformstl/synch/spin_mutex.hpp>
#include <platformstl/synch/thread_mutex.hpp>
#include <platformstl/synch/process_mutex.hpp>

#include <stlsoft/synch/null_mutex.hpp>
#include <stlsoft/synch/lock_scope.hpp>
#include <stlsoft/smartptr/scoped_handle.hpp>

#include <conio.h>      // _kbhit()
#include <iostream>     // std::cout, std::endl
#include <iomanip>      // std::hex

// choose one of the synchronisation methods below:

//typedef stlsoft::null_mutex          mutex_t;
typedef platformstl::spin_mutex      mutex_t;
//typedef platformstl::thread_mutex    mutex_t;
//typedef platformstl::process_mutex   mutex_t;

typedef stlsoft::lock_scope< mutex_t > lock_scope_t;

volatile int x1 = 0xA500, x2 = 0x00A5, x3 = 0x5A00, x4 = 0x005A;

struct shared
{
   shared() : m_x(0) { ; }

   //                                          -- critical section --
   void produce() { lock_scope_t lock( m_mutex ); m_x = x1; m_x += x2;  }
   void consume() { lock_scope_t lock( m_mutex ); m_x = x3; m_x += x4;  }
   int      get() { lock_scope_t lock( m_mutex ); return m_x;           }

private:
   int     m_x;      // the protected object
   mutex_t m_mutex;  // the mutual exclusion object
};

void run_threads();

int main()
{
   std::cout << "Producer-consumer example using critical section\n" << std::endl;

   run_threads();
}

DWORD WINAPI producer( LPVOID arg )
{
   std::cerr << "producer running" << std::endl;

   while ( true )
   {
      static_cast< shared* >( arg )->produce();
   }
   return 0;
}

DWORD WINAPI consumer( LPVOID arg )
{
   std::cerr << "consumer running" << std::endl;

   while ( true )
```

```
   {
      static_cast< shared* >( arg )->consume();
   }
   return 0;
}

typedef DWORD (WINAPI *function_t)( LPVOID );

HANDLE make_thread( const char *msg, function_t f, shared *object );

void run_threads()
{
   shared shared_object;

   stlsoft::scoped_handle< HANDLE > producer_thread( make_thread( "producer", producer, &shared_object  ), ::CloseHandle );
   stlsoft::scoped_handle< HANDLE > consumer_thread( make_thread( "consumer", consumer, &shared_object  ), ::CloseHandle );

   std::cerr << "\nPress a key to stop...\n" << std::endl;

   ::ResumeThread( producer_thread.get() );
   ::ResumeThread( consumer_thread.get() ); ::Sleep( 10 );

   // wait for key pressed; eat character:
   while ( ! ::_kbhit() )
   {
      int x = shared_object.get();

      if ( x != ( x1 + x2 ) && x != ( x3 + x4 ) )
      {
         std::cout << std::hex << x << ' ';
      }
   }
   (void) ::_getch();

   // stop and remove threads:
   std::cerr << "\nTerminating..." << std::endl;
}

HANDLE make_thread( const char *msg, function_t function, shared *object )
{
   std::cerr << "creating thread for " << msg << std::endl;

   DWORD id;

   return CreateThread
   ( NULL               // pointer to thread security attributes
   , 0                  // initial thread stack size, in bytes
   , function           // pointer to thread function
   , object             // argument for new thread
   , CREATE_SUSPENDED   // creation flags
   , &id                // pointer to returned thread identifier
   );
}
```

PlatformSTL System Library (environment map)

```
#include <platformstl/system/environment_map.hpp>

#include <algorithm>    // std::count_if, std::distance
#include <iostream>     // std::cout, std::endl

using platformstl::environment_map;
using stlsoft::c_str_ptr;

struct starts_with : public std::unary_function< environment_map::value_type, bool >
{
   explicit starts_with( char chr ) : m_chr( chr ) { ; }

   bool operator()( environment_map::value_type const& v ) const { return m_chr == *c_str_ptr( v.first ); }

   const char m_chr;
};

int main()
{
   try
   {
      environment_map map;

      std::cout << "\nThe current environment contains "    <<
                   std::distance( map.begin(), map.end() ) << " entries, of which " <<
                   std::count_if( map.begin(), map.end(), starts_with( 'S' ) )      <<
                                                            " start with 'S'."    << std::endl;
```

```
      std::string content;
      if ( map.lookup( "STLSoft", content ) )
      {
         std::cout << "\nSTLSoft is defined with value: '" << content << "'" << std::endl;
      }
      else
      {
         std::cout << "\nSTLSoft is not defined" << std::endl;
      }

      std::cout << "\n       Computer name: " << ( map[ "computername"         ] ) <<
                   "\n    Operating system: " << ( map[ "os"                   ] ) <<
                   "\nProcessor identifier: " << ( map[ "processor_identifier" ] ) << std::endl;
   }
   catch ( std::exception const& e )
   {
      std::cerr << "Exception: " << e.what() << std::endl;
   }
}
```

RangeLib Project

RangeLib (range accumulate)

```
#include <rangelib/algorithms.hpp>
#include <rangelib/integral_range.hpp>

#include <iterator>     // std::ostream_iterator<>
#include <iostream>     // std::copy, std::cout, std::endl

int main()
{
   // create range of integer values [ -10, +10 ), in increments of 2:
   rangelib::integral_range< int >  r( -10, +10  , 2 );

   // print the range:
   r_copy( r, std::ostream_iterator< int >( std::cout, " " ) );

   // calculate the sum:
   int sum = r_accumulate( r, 0 );

   std::cout << "--- sum of range [-10, +10), step 2 must be -10: " << sum << std::endl;
}
```

STLSoft Project

STLSoft (contract programming) Remember to compile for debug mode by defining _DEBUG (–D_DEBUG) and linking with the appropriate libraries (e.g. option –MDd for Visual C++). Note that you also can define your own assert function for STLSoft. See **STLSoft (custom assert)**. See also [**Dawson**] on comparing floating point numbers.

```
#include <stlsoft/stlsoft.h>
#include <stlsoft/meta/is_numeric_type.hpp>

#include <iostream>     // std::cout, std::endl

template< typename T > void use( T x ) { }

inline bool equal( int    x, int    y ) { return x == y; }
inline bool equal( long   x, long   y ) { return x == y; }
inline bool equal( double x, double y ) { return false;  }  // not implemented !!

template < typename T >
T my_sqrt( T x )
{
   typedef T value_type;

   STLSOFT_STATIC_ASSERT( stlsoft::is_numeric_type< value_type >::value );

   STLSOFT_MESSAGE_ASSERT( "Precondition violation: x < 0", x >= 0 );

   value_type result( x / 2 ); // bad implementation !!

   STLSOFT_MESSAGE_ASSERT(
      "Postcondition violation: result * result != x", equal( x, result * result ) );

   return result;
}

struct S{}; S s;

int main()
```

```
{
// use ( my_sqrt( s ) );    // not a numric type: fires static assertion

   use( my_sqrt( -1   ) );  // less than zero: fires precondition assertion
   use( my_sqrt(  9L  ) );  // bad implementation, bad result: fires postcondition assertion
   use( my_sqrt(  5.0 ) );  // bad implementation, and equal needs proper implementation for
                            //   floating point: also fires postcondition assertion
}

// cl -EHsc -MDd -D_DEBUG -I"%STLSOFT%/include" stlsoft_contract.cpp
```

STLSoft (custom assert) Compile with:

```
prompt>g++ -I. -I"%STLSOFT%/include" \
               -D"_STLSOFT_CUSTOM_ASSERT(_x)=MY_ASSERT(_x)" \
               -D"_STLSOFT_CUSTOM_ASSERT_INCLUDE=\"stlsoft_custom_assert.h\"" stlsoft_custom_assert.cpp
```

Defining the _STLSOFT_CUSTOM_ASSERT... macros ensures that file stlsoft_custom_assert.h is included at the proper location in STLSoft.

```
// stlsoft_custom_assert.h

#define _STLSOFT_CUSTOM_ASSERT( _x )     MY_ASSERT( _x )
#define _STLSOFT_CUSTOM_ASSERT_INCLUDE  "stlsoft_custom_assert.h"

#define MY_ASSERT( _x ) ((void)((!(_x)) ? (_my_assert(__FILE__, __LINE__, #_x),1) : 0))

// declaration:
void _my_assert( char const* file, int line, char const* expression );
```

Note that MYASSERT() does not depend on _DEBUG being defined or not.

```
// first define custom assert macros, then include stlsoft.h;
// a '' #include "stlsoft_custom_assert.h" '' is in effect at the proper
// location in STLSoft, due to the following commandline defines:
//    -D"_STLSOFT_CUSTOM_ASSERT(_x)=... and
//    -D"_STLSOFT_CUSTOM_ASSERT_INCLUDE=...

#include <stlsoft/stlsoft.h>

#include <iostream>     // std::cout, std::endl

// implementation:
void _my_assert( char const* file, int line, char const* expression )
{
   std::cerr << "An unfortunate but foreseen error occurred, details follow:\n\t" <<
                file << ", line " << line << ": " << expression << std::endl <<
                "The program will be stopped now." << std::endl;
   ::exit( 1 );
}

int main()
{
   STLSOFT_ASSERT( "Show Message" && 0 );
}
```

STLSoft (shims) Adapted from [**Wilson 2003b**, listing 10]. [Type tunneling]

```
#include <stlsoft/shims/access/string.hpp>

#include <iostream>     // std::cout, std::endl

#define MY_MESSAGE_ASSERT( expression, message ) \
   ((void) ((!(expression)) ? \
    (my_message_assert( __FILE__, __LINE__, #expression, message ), 1) : (0) ))

template < typename S1, typename S2 >
inline void my_message_assert( const char* file, int line, S1 const& expession, S2 const& message )
{
   using stlsoft::c_str_ptr;

   std::cout << file << ", " << line << ": " <<
                c_str_ptr( message ) << " (" << c_str_ptr( expession ) << ")" << std::endl;
}

int main()
{
   std::string action_string( "act as what?" );

   MY_MESSAGE_ASSERT( 123 == (1 + 2 + 3), "think as a child" );
```

```
   MY_MESSAGE_ASSERT( 456 == (4 + 5 + 6),  action_string     ); // note: no .c_str()
}
```

STLSoft Collections Library (array view)

```
#include <stlsoft/collections/array_view.hpp>

#include <algorithm>    // std::count_if
#include <iostream>     // std::cout, std::endl

typedef int number_t;

bool is_even( number_t n ) { return 0 == n % 2; }

int main()
{
   number_t numbers[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, };

   stlsoft::array_view< number_t > view( numbers );

   std::cout << "# even: " << std::count_if( view.begin(), view.end(), is_even );
}
```

STLSoft Containers Library (fixed 2d array)

```
#include <stlsoft/containers/fixed_array.hpp>

#include <algorithm>    // std::copy
#include <iterator>     // std::ostream_iterator
#include <iostream>     // std::cout, std::endl

using stlsoft::fixed_array_2d;

int main()
{
   typedef char            element_t;
   typedef fixed_array_2d< element_t > array_2d_t;

   array_2d_t array_2d( 3, 4, '-' );

   array_2d.at( 0, 0 ) = '0';  array_2d.at( 0, 3 ) = '3';
   array_2d.at( 1, 0 ) = '4';  array_2d.at( 1, 3 ) = '7';
   array_2d.at( 2, 0 ) = '8';  array_2d.at( 2, 3 ) = 'B';

   for ( size_t i0 = 0; i0 < array_2d.size() / array_2d.dimension1(); ++i0 )
   {
      array_2d_t::dimension_element_type const& row = array_2d.at( i0 );

      std::copy
      ( row.begin()
      , row.end()
      , std::ostream_iterator< array_2d_t::value_type >( std::cout, ", " )
      );
      std::cout << std::endl;
   }
}
```

STLSoft Functional Library ()

STLSoft Iterators Library Library (ostream_iterator)

```
#include <stlsoft/iterators/ostream_iterator.hpp>

#include <iostream>     // std::cout, std::endl

int main()
{
   int array[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, };

   std::cout << "Elements:" << std::endl;

   // note presence of both prefix and suffix in output iterator:
   std::copy
   ( array
   , array + STLSOFT_NUM_ELEMENTS( array )
   , stlsoft::ostream_iterator< int >( std::cout, "\t", "\n" )
   );
```

```
}
```

STLSoft Memory Library (auto buffer) Adapted from STLSoft documentation.

```
#include <stlsoft/memory/auto_buffer.hpp>

#include <iostream>     // std::cout, std::endl

int main()
{
   try
   {
      stlsoft::auto_buffer< char, 64 >  buff( 0 );

      // resize within the bounds of the internal buffer:
      buff.resize( 10 );

      ::memset( &buff[0], 1, buff.size() );

      // resize outside the bounds of the internal buffer, and go to the heap:
      buff.resize( 100 );

      ::memset( &buff[0], 2, buff.size() );
   }
   catch ( std::exception const& e )
   {
      std::cerr << "Error: " << e.what() << std::endl;
   }
   catch (...)
   {
      std::cerr << "Unknown error" << std::endl;
   }
}
```

STLSoft Properties Library (field and method properties)

```
// compiles with VC8, not with VC6 or GCC 3.4.1

#include <stlsoft/properties/field_properties.hpp>
#include <stlsoft/properties/method_properties.hpp>

#include <iostream>     // std::cout, std::endl
#include <string>       // std::string, output

class Character
{
public:
   typedef std::string  name_t;
   typedef int          year_t;

   Character( name_t name, year_t year )
   : name( name )
   , year( year )
   { ; }

   year_t get_age() const { return 2008 - year; } // simplistic

   // two read-only properties:
   //                    internal type | external type | class     | property name
   stlsoft::field_property_get < name_t, name_t const &, Character > name;
   stlsoft::field_property_get < year_t,         year_t, Character > year;

   //                               return type | class    | read fn | property name
   STLSOFT_METHOD_PROPERTY_GET_EXTERNAL( year_t, Character, get_age , age           );
};

int main()
{
   Character dduck( "donald duck", 1934 );

//   dduck.name = "dagobert duck";  // does not compile !

   std::cout << "dduck.name: " << dduck.name << std::endl <<
                "dduck.year: " << dduck.year << std::endl <<
                " dduck.age: " << dduck.age  << " (in 2008 that is)" << std::endl;
}

// cl -nologo -W3 -EHsc -I"%STLSoft%/include" stlsoft_properties.cpp
```

STLSoft Smart Pointer Library (scoped handle) Adapted from [**Wilson 2007**, pp.107–108].

```
#include <stlsoft/smartptr/scoped_handle.hpp>

#include <io.h>         // _open, _close
#include <fcntl.h>      // _O_RDONLY
#include <memory.h>     // memset
#include <stdio.h>      // fopen, fclose
#include <stdlib.h>     // malloc, free; already included by STLSoft
#include <winsock2.h>   // WSAStartup, WSACleanup

int main()
{
   void* mp = ::malloc(100);

   if ( NULL != mp )
   {
      ::stlsoft::scoped_handle< void* > h_mem( mp, ::free );

      ::memset( h_mem.get(), 0, 100 );

   } // free() called here


   FILE* fp = ::fopen( "file.ext", "r" );

   if ( NULL != fp )
   {
      ::stlsoft::scoped_handle< FILE* > h_file( fp, ::fclose );

      // use h_file.get() here...

   } // fclose() called here


   // handle non-null 'null' values:
   int fd = ::_open( "file.ext", _O_RDONLY );

   if ( -1 != fd )
   {
      ::stlsoft::scoped_handle< int > h_fd ( fd, ::_close, -1 ); // note -1

      // use h_fd.get() here...

   }  // close() called here


   // also handles stdcall functions


   // also can scope function taking no parameters, like:
   WSADATA wsadata;
   if ( 0 != ::WSAStartup( 0x202, &wsadata ) )
   {
      ::stlsoft::scoped_handle< void > h_wsa ( ::WSACleanup ); // note void

      // use WinSock API, h_wsa.get()

   } // WSACleanup() clled here
}
```

STLSoft Smart Pointer Library (shared pointer) Note that shared_ptr is defined in [**Technical Report 1**] as std::tr1::shared_ptr and will become part of C++ in the next standard [**C++0x**].

```
#include <stlsoft/smartptr/shared_ptr.hpp>

#include <stdexcept> // std::logic_error, std::exception
#include <iostream>  // std::cout, std::endl

struct logger
{
    logger( const char* name ) : m_name( name )
             { std::cerr << m_name << ": constructing" << std::endl; }
   ~logger() { std::cerr << m_name << ": destructing"  << std::endl; }

   const char* m_name;
};

int main()
{
   stlsoft::shared_ptr< logger > p( new logger( "p" ) );

   try
   {
      stlsoft::shared_ptr< logger > q( p );  // share p
```

```
      stlsoft::shared_ptr< logger > r( new logger( "r" ) );

      throw std::logic_error( "operation failed" );

      std::cerr << "end of try block: " << std::endl;
   }
   catch ( std::exception const& e )
   {
      std::cerr << "exception: " << e.what() << std::endl;
   }

   std::cerr << "end of function." << std::endl;
}
```

STLSoft String Library (string tokeniser)

```
#include <stlsoft/string/string_tokeniser.hpp>

#include <iostream>     // std::cout, std::endl

int main()
{
}
```

STLSoft Template Meta Programming Library (type traits)

```
#include <stlsoft/meta/add_qualifier.hpp>
#include <stlsoft/meta/is_compound_type.hpp>
#include <stlsoft/meta/is_const_type.hpp>
#include <stlsoft/meta/is_numeric_type.hpp>
#include <stlsoft/meta/is_pointer_type.hpp>
#include <stlsoft/meta/is_same_type.hpp>
#include <stlsoft/meta/detector/has_value_type.hpp>
#include <stlsoft/meta/select_first_type_if.hpp>

#include <iostream>     // std::cout, std::endl

using namespace stlsoft;

struct Struct {};

class Class { typedef int value_type; };

#define DO( cmd, result ) \
   std::cout << #cmd << ": " << cmd << " --- " << result << std::endl;

int main()
{
   DO( is_numeric_type < int        >::value, "yes" );
   DO( is_numeric_type < Struct     >::value, "no" );

   DO( is_pointer_type < int const* >::value, "yes" );
   DO( is_pointer_type < int*       >::value, "yes" );
   DO( is_pointer_type < int        >::value, "no" );

   DO( is_const_type   < int const  >::value, "yes" );
   DO( is_const_type   < int        >::value, "no" );

   DO( is_compound_type< Struct     >::value, "yes" );
   DO( is_compound_type< int        >::value, "no" );

   DO( ( is_same_type  < int, int   >::value ), "yes" );
   DO( ( is_same_type  < int, int&  >::value ), "no" );

   DO( has_value_type  < int         >::value, "no" );
   DO( has_value_type  < Struct      >::value, "no" );
   DO( has_value_type  < Class       >::value, "yes" );

   DO( ( is_pointer_type< select_first_type_if< int*, int, true  >::type >::value ), "yes" );
   DO( ( is_pointer_type< select_first_type_if< int*, int, false >::type >::value ), "no" );
}
```

STLSoft Template Meta Programming Library (parameter type) Note that the usage of the is_reference class template in this example requires a standards conforming compiler like VC8, GNU C 4.

```
#include <stlsoft/meta/add_qualifier.hpp>
#include <stlsoft/meta/is_numeric_type.hpp>
#include <stlsoft/meta/is_pointer_type.hpp>
#include <stlsoft/meta/select_first_type_if.hpp>
#include <stlsoft/meta/yesno.hpp>
```

```
#include <iostream>     // std::cout, std::endl
#include <typeinfo>     // typeid

template < class T >
struct identity
{
   typedef T type;
};

template < class T, class F, bool C > struct eval_if
   : stlsoft::select_first_type_if< T, F, C >::type
{};

template < class T > struct is_reference : stlsoft::value_to_yesno_type<false> {};
template < class T > struct is_reference< T& > : stlsoft::value_to_yesno_type<true>{};

// using lazy evaluation:
template < class T > struct parameter_type
   : eval_if
     < identity<T>
     , stlsoft::add_const_ref< T >
     , stlsoft::is_numeric_type<T>::value ||
       stlsoft::is_pointer_type<T>::value ||
       is_reference<T>::value
     > // no ::type here
{};

template < class T >
struct holder
{
   holder( typename parameter_type< T >::type x )
   : m_x( x )
   {
      if ( is_reference< typename parameter_type< T >::type >::value )
      {
         std::cout << "by-reference type (" << typeid( x ).name() << ")";
      }
      else
      {
         std::cout << "by-value type ("  << typeid( x ).name() << ")";
      }
   }

   T m_x;
};

struct S { };

#define DO( cmd, result ) \
   std::cout << #cmd << ": "; cmd; std::cout << " --- " << result << std::endl;

int main()
{
   int x; int& rx(  x ); int* px( &x ); S s;

   DO( { holder< int  > h( x  ); }, "by-value" );
   DO( { holder< int* > h( px ); }, "by-value" );
   DO( { holder< int& > h( rx ); }, "by-reference" );
   DO( { holder<  S   > h( s  ); }, "by-reference" );
}
```

STLSoft Template Meta Programming Library (parameter type, TR1) Note that the usage of the is_reference class template in this example requires a standards conforming compiler like VC8, GNU C 4. See also [**Abrahams 2004**].

```
#include <stlsoft/meta/add_qualifier.hpp>
#include <stlsoft/meta/is_numeric_type.hpp>

#include <iostream>     // std::cout, std::endl
#include <typeinfo>     // typeid

template < class T >
struct identity
{
   typedef T type;
};

template< bool x > struct bool_
{
   static bool const value = x;
   typedef bool_< x > type;
   typedef bool value_type;
   operator bool() const { return x; }
```

```
};

typedef bool_< false > false_;
typedef bool_< true  > true_;

template < class C, class T, class F > struct if_
{
   typedef T type;
};

template < class T, class F > struct if_< false_, T, F >
{
   typedef F type;
};

template < class C, class T, class F > struct eval_if
   : if_< typename C::type, T, F >::type
{};

template < class A, class B = false_, class C = false_, class D = false_ > struct or_
   : bool_< 0 != ( A::value || B::value || C::value || D::value ) >
{};

template < class T > struct is_arithmetic
   : bool_< 0 != stlsoft::is_numeric_type<T>::value > {};

template < class T > struct is_enum : false_ {}; // not implemented
template < class T > struct is_member_pointer : false_ {}; // not implemented

template < class T > struct is_reference : false_ {};
template < class T > struct is_reference< T& > : true_ {};

template < class T > struct is_pointer : false_ {};
template < class T > struct is_pointer< T* > : true_ {};

template < class T > struct is_scalar
   : or_< is_arithmetic<T>
        , is_enum<T>
        , is_pointer<T>
        , is_member_pointer<T>
        >
{};

// using lazy evaluation:
template < class T > struct parameter_type
   : eval_if
      < or_< is_scalar< T >, is_reference< T > >
      , identity< T >
      , stlsoft::add_const_ref< T >
      > // no ::type here
{};

template < class T >
struct holder
{
   holder( typename parameter_type< T >::type x )
   : m_x( x )
   {
      if ( is_reference< typename parameter_type< T >::type >::value )
      {
         std::cout << "by-reference type (" << typeid( x ).name() << ")";
      }
      else
      {
         std::cout << "by-value type ("  << typeid( x ).name() << ")";
      }
   }

   T m_x;
};

struct S { };

#define DO( cmd, result ) \
   std::cout << #cmd << ": " << cmd << " --- " << result << std::endl;

#define BE( cmd, result ) \
   std::cout << #cmd << ": "; cmd; std::cout << " --- " << result << std::endl;

int main()
{
   DO( is_scalar< int >::value, "yes" );
   DO( is_scalar< int& >::value, "yes" );
   DO( is_scalar<  S  >::value, "no"  );
```

```
   DO( is_arithmetic< int >::value, "yes" );
   DO( is_arithmetic< int& >::value, "yes" );
   DO( is_arithmetic<  S  >::value, "no" );

   DO( is_reference<int >::value, "no"  );
   DO( is_reference<int&>::value, "yes" );
   DO( is_reference<int const&>::value, "yes" );

   DO( is_pointer<int >::value, "no"  );
   DO( is_pointer<int*>::value, "yes" );
   DO( is_pointer<int const*>::value, "yes" );

   int x; int& rx(  x ); int* px( &x ); S s;

   BE( { holder< int  > h( x  ); }, "by-value" );
   BE( { holder< int* > h( px ); }, "by-value" );
   BE( { holder< int& > h( rx ); }, "by-reference" );
   BE( { holder<  S   > h( s  ); }, "by-reference" );
}
```

UNIXSTL Project

UNIXSTL Filesystem Library (glob sequence)

```
#include <unixstl/filesystem/glob_sequence.hpp>

#include <iostream>     // std::cout, std::endl

using unixstl::glob_sequence;

void print_name( char const* name )
{
   std::cout << name << std::endl;
}

int main()
{
   unixstl::glob_sequence files( ".", "*.cpp",  glob_sequence::files );

   std::for_each( files.begin(), files.end(), print_name );
}
```

WinSTL Project

WinSTL DL Library (dl call) Note that the DLL and functionname strings can be any string type for which a c_str_ptr shim is defined. The calling convention can be specified as a prefix with the functionname string.

- cdecl: C: or cdecl: or nothing
- fastcall : F: or fastcall:
- stdcall: S: or stdcall:

See also **PlatformSTL DL Library (module)**.

```
#include <winstl/dl/dl_call.hpp>

#include <iostream>     // std::cout, std::endl
#include <string>       // std::string

std::string get_system_dir()
{
   typedef DWORD return_t;

   TCHAR dir[ 1 + _MAX_PATH ];

   return_t length = winstl::dl_call< return_t >
   ( "KERNEL32.DLL"              // DLL
   , "S:GetSystemDirectoryA"     // S: stdcall
   , &dir[0]                     // parameter 1
   , STLSOFT_NUM_ELEMENTS( dir ) // parameter 2
   );

   return 0 == length ? "[error]" : dir;
}

int main()
{
   try
   {
      std::cout << "System directory: " << get_system_dir() << std::endl;
   }
```

```
   catch ( std::exception const& e)
   {
      std::cerr << "Error: " << e.what() << std::endl;
   }
}
```

WinSTL Filesystem Library (findfile sequence)

```
#include <winstl/filesystem/findfile_sequence.hpp>

#include <algorithm>    // std::for_each
#include <iterator>     // std::ostream_iterator
#include <iostream>     // std::cout, std::endl

using winstl::findfile_sequence;

void print_name( char const* name )
{
   std::cout << name << std::endl;
}

int main()
{
   findfile_sequence files( ".", "*.cpp|*.o", '|', findfile_sequence::files );

   std::for_each( files.begin(), files.end(), print_name );
}
```

WinSTL System Library (pid sequence) See also **WinSTL System Library (process module sequence)**

```
#include <winstl/system/pid_sequence.hpp>

#include <exception>    // std::exception
#include <iterator>     // std::ostream_iterator
#include <iostream>     // std::cout, std::endl

int main()
{
   using winstl::pid_sequence;

   pid_sequence pids( pid_sequence::elideIdle | pid_sequence::elideSystem | pid_sequence::sort );

   std::copy
   ( pids.begin()
   , pids.end()
   , std::ostream_iterator< pid_sequence::value_type >( std::cout, "\n" )
   );
}
```

WinSTL System Library (process module sequence) Adapted from [**Wilson 2007**, pp.206–210]. See also **WinSTL System Library (pid sequence)**

```
#include <winstl/system/pid_sequence.hpp>
#include <winstl/system/process_module_sequence.hpp>

#include <stlsoft/smartptr/scoped_handle.hpp>
#include <stlsoft/string/static_string.hpp>

#include <exception> // std::exception
#include <iostream>  // std::cout, std::endl

#include <psapi.h>   // ::GetModuleFileNameEx()

using winstl::pid_sequence;
using winstl::process_module_sequence;

stlsoft::basic_static_string< char, _MAX_PATH >
get_module_path( HANDLE hProcess, HMODULE hModule)
{
   stlsoft::basic_static_string< char, _MAX_PATH > path( "", _MAX_PATH );

   DWORD length = ::GetModuleFileNameEx( hProcess, hModule, &path[0], _MAX_PATH );

   if ( 0 == length )
   {
      throw winstl::windows_exception( "Cannot determine module path", ::GetLastError() );
   }
   path.resize( length );

   return path;
```

```
}

struct do_module
{
   explicit do_module( HANDLE hProcess ) : m_hProcess( hProcess ) { ; }

   void operator()( process_module_sequence::value_type const& hModule ) const
   {
      std::cout << "  " << get_module_path( m_hProcess, hModule ) << ": " << std::endl;
   }

   HANDLE m_hProcess;
};

void do_pid( pid_sequence::value_type const& pid )
{
   HANDLE hProcess = ::OpenProcess
   ( PROCESS_QUERY_INFORMATION | PROCESS_VM_READ
   , false
   , pid
   );

   if ( NULL == hProcess )
   {
      throw winstl::windows_exception( "Cannot open process", ::GetLastError() );
   }

   ::stlsoft::scoped_handle< HANDLE > scoped( hProcess, ::CloseHandle );

   std::cout << std::endl << get_module_path( hProcess, NULL ) << ": " << std::endl;

   process_module_sequence modules( hProcess );

   std::for_each( modules.begin(), modules.end(), do_module( hProcess ) );
}

int main()
{
   try
   {
      pid_sequence pids( pid_sequence::elideIdle | pid_sequence::elideSystem | pid_sequence::sort );

      std::for_each( pids.begin(), pids.end(), do_pid );
   }
   catch ( std::exception const& e )
   {
      std::cerr << "Error: " << e.what() << std::endl;
   }
}

// g++ -Wall -I"%STLSOFT%/include" -L$(MINGW)/lib winstl_process_module_sequence.cpp -l psapi -o winstl_process_module_sequence
```

WinSTL System Library (system info)

```
#include <winstl/system/system_info.hpp>

#include <iostream>     // std::cout, std::endl

int main()
{
   using winstl::system_info;

   std::cout << "\n            System info"   <<
                "\n   number of processors: " << system_info::number_of_processors() <<
                "\n              page size: " << system_info::page_size () <<
                "\n allocation granularity: " << system_info::allocation_granularity () << std::endl;
}
```

WinSTL Windows Clipboard Library (clipboard scope) Adapted from STLSoft documentation.

```
#include <winstl/clipboard/clipboard_scope.hpp>

#include <algorithm>    // std::copy
#include <iostream>     // std::cout, std::endl

int main()
{
   try
   {
      // set the data on the clipboard:
      {
         winstl::clipboard_scope  scope;
```

```
         scope.set_data( "Text from clipboard_scope demo." );
      }

      // as long as no other thread/process changes the clipboard contents
      // in the meanwhile, this can then be read back, as follows:
      {
         winstl::clipboard_scope   scope;
         char const                *cstr;

         scope.get_data( cstr );

         std::cout << "Clipboard data: " << cstr << std::endl;
      }
   }
   catch ( std::exception const& e )
   {
      std::cerr << "Exception: " << e.what() << std::endl;
   }
}

// cl -W3 -EHsc -I%STLSOFT%/include winstl_clipboard_scope.cpp user32.lib
```

WinSTL Windows Clipboard Library (clipboard format sequence)

```
#include <winstl/clipboard/clipboard_format_sequence.hpp>

#include <algorithm>    // std::copy
#include <iterator>     // std::ostream_iterator
#include <iostream>     // std::cout, std::endl

int main()
{
   typedef winstl::clipboard_format_sequence::value_type value_type;

   winstl::clipboard_format_sequence formats;

   std::copy
   ( formats.begin()
   , formats.end()
   , std::ostream_iterator< value_type >( std::cout, "\n" )
   );
}

// cl -W3 -EHsc -I%STLSOFT%/include winstl_clipboard_format_sequence.cpp user32.lib
```

WinSTL Windows Control Panel Library (control panel applet)

Adapted from STLSoft documentation.

```
// Note: system_directory.hpp first, otherwise VC6 gives internal compiler error (C1001)
#include <winstl/system/system_directory.hpp>
#include <winstl/control_panel/applet_module.hpp>
#include <winstl/filesystem/findfile_sequence.hpp>

#include <algorithm>    // std::for_each
#include <iostream>     // std::cout, std::endl

void print_applet( winstl::applet_module::value_type const& applet )
{
   std::cout << " applet index: " << applet.get_index()       << std::endl <<
                "         name: " << applet.get_name()        << std::endl <<
                "  description: " << applet.get_description() << std::endl;
}

void do_file( winstl::findfile_sequence::value_type const& file )
{
   winstl::applet_module module( file, winstl::applet_module::dontExpectNonZeroInit );

   std::cout << "\n         path: " << module.get_path() << std::endl;

   std::for_each( module.begin(), module.end(), print_applet );
}

int main()
{
   try
   {
      winstl::system_directory   sysDir;
      winstl::findfile_sequence  files( sysDir, "*.cpl", winstl::findfile_sequence::files );

      std::for_each( files.begin(), files.end(), do_file );
   }
   catch ( std::exception const& e )
```

```
   {
      std::cerr << "Exception: " << e.what() << std::endl;
   }
}

// cl -W3 -EHsc -I%STLSOFT%/include winstl_control_panel_applet.cpp user32.lib
```

WinSTL Windows Registry Library (registry value sequence)

```
#include <winstl/registry/reg_value_sequence.hpp>

#include <iostream>  // std::cout, std::endl

void print_value( winstl::reg_value const& v )
{
   std::cout << v.name() << ':' << v.value_sz() << std::endl;
}

int main()
{
   winstl::reg_value_sequence values( HKEY_CURRENT_USER, "Environment" );

   std::for_each( values.begin(), values.end(), print_value );
}
```

WinSTL Windows Shell Library (browse for folder/file)

Adapted from STLSoft documentation. See MSDN for CSIDL identifiers and BROWSEINFO structure flags .

```
#include <stlsoft/smartptr/scoped_handle.hpp>   // stlsoft::scoped_handle<>
#include <winstl/error/error_desc.hpp>          // winstl::error_desc()
#include <winstl/shell/browse_for_folder.hpp>   // winstl::browse_for_folder()
#include <winstl/shell/memory/functions.h>      // winstl::SHMemFree()

#include <iostream>     // std::cout, std::endl

#include <stdlib.h>     // EXIT_FAILURE, EXIT_SUCCESS

int main()
{
   try
   {
      CHAR displayName[ _MAX_PATH + 1 ];

      LPITEMIDLIST  iil;
      HRESULT hresult = ::SHGetSpecialFolderLocation
      ( NULL            // hwndOwner
      , CSIDL_DRIVES    // see msdn
      , &iil            // pointer to an item identifier list (PIDL) specifying the folder's
      );                // location relative to the root of the namespace (the desktop)

      if ( FAILED( hresult ) )
      {
         std::cerr << "Could not get the My Computer special folder location: " <<
                      winstl::error_desc( hresult ).c_str() << std::endl;
      }
      else
      {
         stlsoft::scoped_handle< void* > iil_( iil, winstl::SHMemFree );

         if ( winstl::browse_for_folder
              ( "Select a file or folder to display"  // window's text
              , displayName                           // path of selected item
              , BIF_BROWSEINCLUDEFILES                // BROWSEINFO structure flags, see msdn
              , iil                                   // browse starting point
              )
         )
         {
            std::cout << "Item selected: '" << displayName << "'" << std::endl;
         }
      }
   }
   catch ( std::exception const& e )
   {
      std::cerr << "Exception: " << e.what() << std::endl;
      return EXIT_FAILURE;
   }
   return EXIT_SUCCESS;
}
```

WinSTL Windows ToolHelp Library (process sequence)

Note entry type is PROCESSENTRY32.

```
#include <winstl/toolhelp/process_sequence.hpp>

#include <exception> // std::exception
#include <iostream>  // std::cout, std::endl

void print_entry( winstl::process_sequence::value_type const& entry )
{
   std::cout << entry.th32ProcessID << ": " << entry.szExeFile << std::endl;
}

int main()
{
   winstl::process_sequence processes;

   std::for_each( processes.begin(), processes.end(), print_entry );
}
```

WTLSTL Project

Sorry, no examples yet.

Handling Exceptions

Note entry type is PROCESSENTRY32.

```
#include <winstl/toolhelp/process_sequence.hpp>

#include <exception> // std::exception
#include <iostream>  // std::cout, std::endl

#include <stdlib.h>  // EXIT_FAILURE, EXIT_SUCCESS

void print_entry( winstl::process_sequence::value_type const& entry )
{
   std::cout << entry.th32ProcessID << ": " << entry.szExeFile << std::endl;
}

int main()
{
   try
   {
      winstl::process_sequence processes;

      std::for_each( processes.begin(), processes.end(), print_entry );

      return EXIT_SUCCESS;
   }
   catch ( std::exception const& e )
   {
      std::cerr << "Error: " << e.what() << std::endl;

      return EXIT_FAILURE;
   }
   catch ( ... )
   {
      std::cerr << "Unknown error" << std::endl;

      return EXIT_FAILURE;
   }
}
```


Application

A. Library Contents

Projects:

- ACESTL
- ATLSTL

- COMSTL
- .netSTL
- InetSTL
- MFCSTL
- PlatformSTL
- RangeLib
- STLSoft
- UNIXSTL
- WinSTL
- WTLSTL

Libraries:

- Collections Library
- Containers Library
- Conversion Library
- DL Library
- Error Library
- File System Library
- Functional Library
- Iterators Library
- Memory Library
- Template Meta-programming Library
- Performance Library
- Properties Library
- Security Library
- Smart Pointers library
- String Library
- Synchronisation Library
- System Library
- Time Library
- Utility Library
- Windows Clipboard Library
- Windows Control Panel Library
- Windows Controls Library
- Windows Registry Library
- Windows Shell Library
- Windows ToolHelp Library

Prev
Library Overview

Next
B. Supported Platforms

B. Supported Platforms

Architecture--OS--Compiler / 32-bit, 64-bit / Unix, Linux, FreeBSD, Mac OS-X, Sun Solaris?, Windows.

The compilers supported by the STLSoft libraries are:

- Borland C++ 5.5, 5.51, 5.6, 5.6.4 & 5.82
- CodePlay VectorC C/C++ 1
- Comeau 4.3.0.1 & 4.3.3
- Digital Mars C/C++ 8.26 and above
- GCC 2.95, 2.96, 3.2, 3.3, 3.4 & 4.0
- Intel C/C++ 6.0, 7.0, 7.1, 8.0, 9.0, 10.0
- Metrowerks 2.4, 3.0 & 3.2 (CodeWarrior 7.0, 8.0 & 9.0)
- SunPro C/C++ 5.9
- Visual C++ 4.2, 5.0, 6.0, 7.0 (.NET), 7.1 (.NET 2003), 8.0 & 9.0
- Watcom C/C++ 11.0, Open Watcom 1.0, 1.1, 1.2, 1.3, 1.4, 1.5

Prev
A. Library Contents

Next
C. Compiling Examples

C. Compiling Examples

The example programs in this guide are compiled using four makefiles:

- **makefile.src** contains lists of source files
- **makefile.u.gcc** to compile with GNU C on Unix
- **makefile.w.gcc** to compile with GNU C on Windows
- **makefile.w.vc** to compile with Visual C++ on Windows

makefile.src contains the source filenames, split into four groups:

- Cross-platform source files for GNU C or Visual C++ (psources)
- Windows source file for GNU C or Visual C++ (wsources)
- Windows source files that require Visual C++ (wsources_vc)
- Unix source files for GNU C (usources)

makefile.src is included by the makefile for the GNU C compiler (`makefile.u.gcc` and `makefile.w.gcc`) and the makefile the the Visual C++ 8 compiler (`makefile.w.vc`)

```
#
# STLSoft Getting Started
#
#./examples/makefile.src - GNU/VC8 makefile with sources for example programs.
#
# $Id: $
#

psources     = platformstl_environment_map.cpp \
               platformstl_path.cpp \
               platformstl_performance_counter.cpp \
               platformstl_readdir_sequence.cpp \
               rangelib_accumulate.cpp \
               stlsoft_array_view.cpp \
               stlsoft_auto_buffer.cpp \
               stlsoft_contract.cpp \
               stlsoft_fixed_array_2d.cpp \
               stlsoft_ostream_iterator.cpp \
               stlsoft_shared_pointer.cpp \
               stlsoft_shims.cpp \
               stlsoft_string_tokeniser.cpp \
               stlsoft_tmp_parameter_type.cpp \
               stlsoft_tmp_parameter_type_tr1.cpp \
               stlsoft_tmp_type_traits.cpp \

wsources     = platformstl_critical_section.cpp \
               platformstl_dl_module.cpp \
               stlsoft_scoped_handle.cpp \
               winstl_dl_call.cpp \
               winstl_findfile_sequence.cpp \
               winstl_pid_sequence.cpp \
               winstl_process_module_sequence.cpp \
               winstl_process_sequence.cpp \
               winstl_process_sequence_exception.cpp \
               winstl_reg_value_sequence.cpp \
               winstl_system_info.cpp \
               winstl_control_panel_applet.cpp \
               winstl_shell_browse_for_folder.cpp

#                platformstl_pipe_client.cpp \
#                platformstl_pipe_server.cpp \

wsources_vc  = inetstl_findfile_sequence.cpp \
               inetstl_ftpdir_sequence.cpp \
               mfcstl_carray_cadaptor.cpp \
               mfcstl_carray_iadaptor.cpp \
               stlsoft_properties.cpp \
               winstl_clipboard_scope.cpp \
               winstl_clipboard_format_sequence.cpp

usources     = unixstl_glob_sequence.cpp

#
# end of file
#
```

makefile.u.gcc is for the GNU make and the GNU C++ compiler on Unix. Usage: `make -f makefile.u.gcc`

```
#
# STLSoft Getting Started
#
# ./examples/makefile.u.gcc - GNU makefile for example programs on Unix.
#
# $Id: $
#
```

```
include makefile.src

pprograms  = $(psources:.cpp=)
uprograms  = $(usources:.cpp=)

CPPFLAGS  += -Wall -I"$(STLSOFT)/include"
LDLIBS    +=

#
# targets:
#
all: platformprog unixprog

platformprog: $(pprograms)

unixprog: $(uprograms)

clean:
        -rm *.bak *.o *.obj

distclean: clean
        -rm $(pprograms) $(uprograms)

realclean: distclean

#
# end of file
#
```

makefile.w.gcc is for the GNU make and the GNU C++ compiler on Windows. Usage: `make -f makefile.w.gcc`

```
#
# STLSoft Getting Started
#
# ./examples/makefile.w.gcc - GNU makefile for example programs on Windows.
#
# $Id: $
#

include makefile.src

pprograms  = $(psources:.cpp=)
wprograms  = $(wsources:.cpp=)

CPPFLAGS  += -Wall -I"$(STLSOFT)/include" -L"$(MINGW)/lib"
LDLIBS    += -lpsapi -lws2_32 # -luser32 -lwininet

#
# targets:
#
all: platformprog winprog

platformprog: $(pprograms)

winprog: $(wprograms)

clean:
        -rm *.bak *.o *.obj

distclean: clean
        -rm *.exe

realclean: distclean

#
# end of file
#
```

makefile.w.vc is for Microsoft nmake and the Visual C++ 8 compiler on Windows. Usage: `nmake -f makefile.w.vc`

```
#
# STLSoft Getting Started
#
# ./examples/makefile.w.vc - VC8 makefile for example programs on Windows.
#
# $Id: $
#

!include makefile.src

pprograms   = $(psources:.cpp=.exe)
wprograms   = $(wsources:.cpp=.exe) $(wsources_vc:.cpp=.exe)
```

```
LDLIBS      = $(LDLIBS) advapi32.lib psapi.lib shell32.lib user32.lib wininet.lib ws2_32.lib
CPPFLAGS    = $(CPPFLAGS) -nologo -W3 -EHsc -I"$(STLSOFT)/include" -D"WINVER=0x0400" \
                -D"_CRT_SECURE_NO_DEPRECATE" -D"_SCL_SECURE_NO_DEPRECATE"$(LDLIBS)

COMPILE_vc_deb_cpp = $(CPP) -MDd -D"_DEBUG"  $(CPPFLAGS) $(LDLIBS)
COMPILE_vc_afx_cpp = $(CPP) -MDd -D"_AFXDLL" $(CPPFLAGS) $(LDLIBS)

#
# targets:
#
all: platformprog winprog

platformprog: $(pprograms)

winprog: $(wprograms)

clean:
        -rm *.bak *.o *.obj

distclean: clean
        -rm *.exe

realclean: distclean

#
# file rules
#
inetstl_findfile_sequence.exe: inetstl_findfile_sequence.cpp
        $(COMPILE_vc_afx_cpp) $*.cpp

inetstl_ftpdir_sequence.exe: inetstl_ftpdir_sequence.cpp
        $(COMPILE_vc_afx_cpp) $*.cpp

mfcstl_carray_cadaptor.exe: mfcstl_carray_cadaptor.cpp
        $(COMPILE_vc_afx_cpp) $*.cpp

mfcstl_carray_iadaptor.exe: mfcstl_carray_iadaptor.cpp
        $(COMPILE_vc_afx_cpp) $*.cpp

stlsoft_contract.exe:stlsoft_contract.cpp
        $(COMPILE_vc_deb_cpp) $*.cpp

#
# end of file
#
```

Prev **Next**
B. Supported Platforms **D. Document License**
STLSoft - Getting Started Guide

D. Document License

To be determined.

Prev **Next**
C. Compiling Examples **E. STLSoft License**
STLSoft - Getting Started Guide

E. STLSoft License

STLSoft uses the BSD-form license, as follows:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name(s) of Matthew Wilson and Synesis Software nor the names of any contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,

WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

References

- Abrahams, David and Aleksey Gurtovoy (2004). C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond . Addison-Wesley. ISBN 0-321-22725-5
- Alexandrescu, Andrei (2001). Modern C++ Design: Generic Programming and Design Patterns Applied . Addison–Wesley. ISBN 0201704315.
- C++0x, http://en.wikipedia.org/w/index.php?title=C%2B%2B0x&oldid=216710331 (last visited Jun. 4, 2008).
- Dawson, Bruce. Comparing floating point numbers (last visited Jun. 6, 2008).
- Koenig, Andrew and Barbara E. Moo (2005), "Templates and Duck Typing ", *C/C++ Users Journal.*
- Technical Report 1, http://en.wikipedia.org/w/index.php?title=Technical_Report_1&oldid=213746450 (last visited Jun. 4, 2008).
- Wilson, Matthew (March 2003). "Adapting Win32 Enumeration APIs to STL Iterator Concepts ", *Windows Developer Magazine* **14** (3)
- Wilson, Matthew (August 2003). "Generalised String Manipulation: Access Shims and Type-tunnelling ", *C/C++ User's Journal*, Volume **21** (8).
- Wilson, Matthew (2004). Imperfect C++: Practical Solutions for Real-Life Programming. Addison-Wesley . ISBN 0-321-22877-4.
- Wilson, Matthew (2007). Extended STL, Volume 1: Collections and Iterators . Addison-Wesley. ISBN 0-321-30550-7.