The NCBI C++ Toolkit Book

The NCBI C++ Toolkit Book

The NCBI C++ Toolkit

26: C Toolkit Resources for C++ Toolkit Users

Last Update: March 10, 2011.

Overview

For certain tasks in the C++ Toolkit environment, it is necessary to use, or at least refer to, material from the NCBI C Toolkit. Here are some links relevant to the C Toolkit:

- C Toolkit Documentation
- C Toolkit Queryable Source Browser

Chapter Outline

The following is an outline of the topics presented in this chapter:

Using NCBI C and C++ Toolkits together

Note: Due to security issues, not all links on this page are accessible by users outside NCBI.

Overview

When using both C and C++ Toolkits together in a single application, it is very important to understand that there are some resources shared between the two. This document describes how to safely use both Toolkits together, and how to gain from their cooperation.

Shared Sources

To maintain a sort of uniformity and ease in source code maintenance, the CONNECT library is the first library of both Toolkits kept the same at the source code level. To provide data interoperability, ASN.1 specifications have to be identical in both Toolkits, too.

CONNECT Library

The CONNECT library is currently the only C code that is kept identical in both Toolkits. The old API of the CONNECT library is still supported by means of a simple wrapper, which is only in the C Toolkit. There are two scripts that perform synchronization between C++ Toolkit and C Toolkit:

sync_c_to_cxx.pl – This script copies the latest changes made in the C Toolkit (which is kept in the CVS repository) to the C++ Toolkit (kept in the Subversion repository). The following files are presently copied: gicache.h and gicache.c. Both are copied from the distrib/network/sybutils/ctlib CVS module to the trunk/c++/src/objtools/data_loaders/genbank/gicache location in the Toolkit repository.

sync_cxx_to_c.pl – This script copies files in the opposite direction: from the C++ Toolkit to the C Toolkit. Most of the files common to both Toolkits are synchronized by this script. Here's the list of C source directories (CVS modules) that are currently copied from Subversion:
- connect
- ctools
- algo/blast/core
- algo/blast/composition_adjustment
- util/tables
- util/creaders

ASN files in the following CVS modules are also synchronized with Subversion:
- network/medarch/client
- network/taxon1/common
- network/id1arch
- network/id2arch
- access
- asn
- biostruc

- biostruc/cdd
- biostruc/cn3d
- tools
- api
- data

ASN.1 Specifications

Unlike the C source files in the CONNECT library, the ASN.1 data specifications are maintained within C Toolkit source structure, and have to be copied over to C++ Toolkit tree whenever they are changed.

However, the internal representations of ASN.1-based objects differ between the two Toolkits. If you need to convert an object from one representation to the other, you can use the template class CAsnConverter<>, defined in ctools/asn_converter.hpp.

Run-Time Resources

The CONNECT library was written for use "as is" in the C Toolkit, but it must also be in the C++ Toolkit tree. Therefore, it cannot directly employ the utility objects offered by the C++ Toolkit such as message logging CNcbiDiag, registry CNcbiRegistry, and MT-locks CRWLock. Instead, these objects were replaced with helper objects coded entirely in C (as tables of function pointers and data).

On the other hand, throughout the code, the CONNECT library refers to predefined objects g_CORE_Log (so called CORE C logger) g_CORE_Registry (CORE C registry), and g_CORE_Lock (CORE C MT-lock), which actually are never initialized by the library, i.e. they are empty objects, which do nothing. It is an application's resposibility to replace these dummies with real working logger, registry, and MT-lock objects. There are two approaches, one for C and another for C++.

C programs can call CORE_SetREG(), CORE_SetLOG(), and CORE_SetLOCK() to set up the registry, logger, and MT-lock (connect/ncbi_util.h must also be included). There are also convenience routines for CORE logger, like CORE_SetLOGFILE(), CORE_SetLOGFILE_NAME(), which facilitate redirecting logging messages to either a C stream (FILE*) or a named file.

In a C++ program, it is necessary to convert native C++ objects into their C equivalents, so that the C++ objects can be used where types LOG, REG or MT_LOCK are expected. This is done using calls declared in connect/ncbi_core_cxx.hpp, as described later in this section.

LOG and CNcbiDiag

The CONNECT library has its own logger, which has to be set by one of the routines declared in connect/ncbi_util.h: CORE_SetLOG(), CORE_SetLOGFILE() etc. On the other hand, the interface defined in connect/ncbi_core_cxx.hpp provides the following C++ function to convert a logging stream of the NCBI C++ Toolkit into a LOG object:

```
LOG LOG_cxx2c (void)
```

This function creates the LOG object on top of the corresponding C++ CNcbiDiag object, and then both C and C++ objects can be manipulated interchangeably, causing exactly the same effect on the underlying logger. Then, the returned C handle LOG can be subsequently used as a CORE C logger by means of CORE_SetLOG(), as in the following nested calls:

```
CORE_SetLOG(LOG_cxx2c());
```

REG and CNcbiRegistry

connect/ncbi_core_cxx.hpp declares the following C++ function to bind C REG object to CNcbiRegistry used in C++ programs built with the use of the NCBI C++ Toolkit:

```
REG REG_cxx2c (CNcbiRegistry* reg, bool pass_ownership = false)
```

Similarly to CORE C logger setting, the returned handle can be used later with CORE_SetREG () declared in connect/ncbi_util.h to set up the global registry object (CORE C registry).

MT_LOCK and CRWLock

There is a function

```
MT_LOCK MT_LOCK_cxx2c (CRWLock* lock, bool pass_ownership = false)
```

declared in connect/ncbi_core_cxx.hpp, which converts an object of class CRWLock into a C object MT_LOCK. The latter can be used as an argument to CORE_SetLOCK() for setting the global CORE C MT-lock, used by a low level code, written in C. Note that passing 0 as the lock pointer will effectively create a new internal CRWLock object, which will then be converted into MT_LOCK and returned. This object gets automatically destroyed when the corresponding MT_LOCK is destroyed. If the pointer to CRWLock is passed a non NULL value then the second argument can specify whether the resulting MT_LOCK acquires the ownership of the lock, thus is able to delete the lock when destructing itself.

CONNECT Library in C++ Code

Setting LOG

To set up the CORE C logger to use the same logging format of messages and destination as used by CNcbiDiag, the following sequence of calls may be used:

```
CORE_SetLOG(LOG_cxx2c());
SetDiagTrace(eDT_Enable);
SetDiagPostLevel(eDiag_Info);
SetDiagPostFlag(eDPF_All);
```

Setting REG

To set the CORE C registry be the same as C++ registry CNcbiRegistry, the following call is necessary:

```
CORE_SetREG(REG_cxx2c(cxxreg, true));
```

Here cxxreg is a CNcbiRegistry registry object created and maintained by a C++ application.

Setting MT-Locking

To set up a CORE lock, which is used throughout the low level code, including places of calls of non-reentrant library calls (if no reentrant counterparts were detected during configure process), one can place the following statement close to the beginning of the program:

```
CORE_SetLOCK(MT_LOCK_cxx2c());
```

The NCBI C++ Toolkit Book

The NCBI C++ Toolkit Book

Note that the use of this call is extremely important in a multi-threaded environment.

Convenience call CONNECT_Init()

Header file connect/ncbi_core_cxx.hpp provides a convenience call, which sets all shared CONNECT-related resources discussed above for an application written within the C++ Toolkit framework (or linked solely against the libraries contained in the Toolkit):

```
void CONNECT_Init(CNcbiRegistry* reg = NULL);
```

The call takes only one argument, an optional pointer to a registry, which is used by the application, and should also be considered by the CONNECT library. No registry will be used if NULL gets passed. The ownership of the registry is passed along. This fact should be noted by an application making extensive use of CONNECT in static classes, i.e. prior to or after main(), because the registry can get deleted before the CONNECT library stops using it. The call also ties CORE C logger to CNcbiDiag, and privately creates a CORE C MT-lock object (on top of CRWLock) for internal synchronization inside the library.

An example of how to use this call can be found in the test program test_ncbi_conn_stream.cpp. It shows how to properly setup CORE C logger, CORE C registry and CORE C MT-lock so they will use the same data in the C and C++ parts of both the CONNECT library and the application code.

Another good source of information is the set of working application examples in src/app/id1_fetch. Note: In the examples, the convenience routine does not change logging levels or disable/enable certain logging properties. If this is desired, the application still has to use separate calls.

C Toolkit diagnostics redirection

In a C/C++ program linked against both NCBI C++ and NCBI C Toolkits the diagnostics messages (if any) generated by either Toolkit are not necessarily directed through same route, which may result in lost or garbled messages. To set the diagnostics destination be the same as CNcbiDiag's one, and thus to guarantee that the messages from both Toolkits will be all stored sequentially and in the order they were generated, there is a call

```
#include <ctools/ctools.h>
void SetupCToolkitErrPost(void);
```

which is put in a specially designated directory ctools providing back links to the C Toolkit from the C++ Toolkit.

CONNECT Library in C Code

The CONNECT library in the C Toolkit has a header connect/ncbi_core_c.h, which serves exactly the same purpose as connect/ncbi_core_cxx.hpp, described previously. It defines an API to convert native Toolkit objects, like logger, registry, and MT-lock into their abstract equivalents, LOG, REG, and MT_LOCK, respectively, which are defined in connect/ncbi_core.h, and subsequently can used by the CONNECT library as CORE C objects.

Briefly, the calls are:

- LOG LOG_c2c (void); Create a logger LOG with all messages sent to it rerouted via the error logging facility used by the C Toolkit.

- REG REG_c2c (const char* conf_file); Build a registry object REG from a named file conf_file. Passing NULL as an argument causes the default Toolkit registry file to be searched for and used.
- MT_LOCK MT_LOCK_c2c (TNlmRWlock lock, int/*bool*/ pass_ownership); Build an MT_LOCK object on top of TNlmRWlock handle. Note that passing NULL effectively creates an internal handle, which is used as an underlying object. Ownership of the original handle can be passed to the resulting MT_LOCK by setting the second argument to a non-zero value. The internally created handle always has its ownership passed along.

Exactly the same way as described in the previous section, all objects, resulting from the above functions, can be used to set up CORE C logger, CORE C registry, and CORE MT-lock of the CONNECT library using the API defined in connect/ncbi_util.h: CORE_SetLOG(), CORE_SetREG(), and CORE_SetLOCK(), respectively.

Convenience call CONNECT_Init()

As an alternative to using per-object settings as shown in the previous paragraph, the following "all-in-one" call is provided:

```
void CONNECT_Init (const char* conf_file);
```

This sets CORE C logger to go via Toolkit default logging facility, causes CORE C registry to be loaded from the named file (or from the Toolkit's default file if conf_file passed NULL), and creates CORE C MT-lock on top of internally created TNlmRWlock handle, the ownership of which is passed to the MT_LOCK.

Note: Again, properties of the logging facility are not affected by this call, i.e. the selection of what gets logged, how, and where, should be controlled by using native C Toolkit's mechanisms defined in ncbierr.h.

Access to the C Toolkit source tree Using CVS

For a detailed description of the CVS utility see the CVS online manual or run the commands "man cvs" or "cvs --help" on your Unix workstation.

CVS Source Code Retrieval for Public Read-only Access

Public access to the public part of the C Toolkit is available via CVS client. To use it, follow exactly the in-house Unix / Mac OS X instructions with two exceptions:

- The CVSROOT env. variable should be set to:
  :pserver:anoncvs@anoncvs.ncbi.nlm.nih.gov:/vault
- Use empty password to login:
  > cvs login
  Logging in to :pserver:anoncvs@anoncvs.ncbi.nlm.nih.gov:/vault
  CVS password: <just press ENTER here>

Public web access is also available via ViewVC.

CVS Source Code Retrieval for In-House Users with Read-Write Access

You must have a CVS account set up prior to using CVS - email svn-admin@ncbi.nlm.nih.gov to get set up.

The C Toolkit CVS repository is available online and may be searched using LXR.

The NCBI C++ Toolkit Book

The NCBI C++ Toolkit Book

Using CVS from Unix or Mac OS X

To set up a CVS client on Unix or Mac OS X:

- Set the CVSROOT environment variable to: :pserver:${LOGNAME}@cvsvault:/src/NCBI/vault.ncbi. Note that for NCBI Unix users, this may already be set if you specified developer for the facilities option in the .ncbi_hints file in your home directory.
- Run the command: cvs login You will be asked for a password (email svn-admin@ncbi.nlm.nih.gov if you need the password). This command will record your login info into ~/.cvspass file so you won't have to login into CVS in the future. Note: You may need to create an empty ~/.cvspass file before logging in as some CVS clients apparently just cannot create it for you. If you get an authorization error, then send e-mail with the errors to cpp-core@ncbi.nlm.nih.gov.
- If you have some other CVS snapshot which was checked out with an old value of CVSROOT, you should commit all your changes first, then delete completely the old snapshot dir and run: cvs checkout to get it with new CVSROOT value.
- Now you are all set and can use all the usual CVS commands.

Note: When you are in a directory that was created with cvs checkout by another person, a local ./CVS/ subdirectory is also created in that directory. In this case, the cvs command ignores the current value of the CVSROOT environment variable and picks up a value from ./CVS/Root file. Here is an example of what this Root file looks like:

```
:pserver:username@cvsvault:/src/NCBI/vault.ncbi
```

Here the *username* is the user name of the person who did the initial CVS checkout in that directory. So CVS picks up the credentials of the user who did the initial check-in and ignores the setting of the CVSROOT environment variable, and therefore the CVS commands that require authorization will fail. There are two possible solutions to this problem:

- Create your own snapshot of this area using the cvs get command.
- Impersonate the user who created the CVS directory by creating in the ~/.cvspass file another string which is a duplicate of the existing one, and in this new string change the username to that of the user who created the directory. This hack will allow you to work with the CVS snapshot of the user who created the directory. However, this type of hack is not recommended for any long term use as you are impersonating another user.

Using CVS from Windows

The preferred CVS client is TortoiseCVS. If this is not installed on your PC, ask PC Systems to have it installed. Your TortoiseCVS installation should include both a CVS command-line client and integration into Windows Explorer.

To use TortoiseCVS as integrated into Windows Explorer:

- Navigate to the directory where you want the source code to be put.
- Right-click and select "CVS Checkout".
- Set the CVSROOT text field to :pserver:%USERNAME%@cvsvault:/src/NCBI/vault.ncbi (where %USERNAME% is replaced with your Windows user name).

- Set the module text field to the portion of the C Toolkit you want to retrieve. If you want the whole Toolkit, use distrib. If you want just one library, for example the CONNECT library, use distrib/connect. There are also non C Toolkit modules (you can see them here). You can work with those as well by using their names instead of distrib (e.g. internal).
- Click OK. If you are asked for a password and don't know what to use, email svn-admin@ncbi.nlm.nih.gov.
- From the context menu (right-click) you can now perform CVS functions, such as updating, committing, tagging, diffing, etc.
- You may also change global preferences (such as external tools) using the Preferences application available from the Start menu.

For command-line use, follow the in-house Unix / Mac OS X instructions with these exceptions:

- Make sure you have your "home" directory set up -- i.e. the environment variables HOMEDRIVE and HOMEPATH should be set. In NCBI, HOMEDRIVE usually set to C:, and HOMEPATH is usually set to something like \Documents and Settings\%USERNAME% (where %USERNAME% is replaced with your Windows user name).
- Create an empty file named .cvspass in your "home" directory.
- The CVS root needs to be specified.
  - Either set an environment variable: %CVSROOT%=:pserver:%USERNAME%@cvsvault:/src/NCBI/vault.ncbi
  - or use a command-line argument for each CVS command: -d :pserver:%USERNAME%@cvsvault:/src/NCBI/vault.ncbi
- Open a command shell, verify the above environment variables are set properly, and execute the command "cvs login". You will be asked for a password (email svn-admin@ncbi.nlm.nih.gov if you need the password). This command will record your login info in the .cvspass file so you won't have to log into CVS in the future. If you get an authorization error, send e-mail with the errors to cpp-core@ncbi.nlm.nih.gov.