An example calling C from Java to get the TAI time

So below is just the C-code (taken shamelessly off the web) to get theTAI, UTC and Monotonic time -

++

```
#include <iostream>
#include <time.h>

long sec(int clock)
{
  struct timespec gettime_now;
  clock_gettime(clock, &gettime_now);
  return gettime_now.tv_nsec;
}

int main()
{
  std::cout << sec(0) << std::endl;       // CLOCK_REALTIME (UTC)
  std::cout << sec(1) << std::endl;       // CLOCK_MONOTONIC
  std::cout << sec(11) << std::endl;      // CLOCK_TAI (TAI)

  return 0;
}
```

++

Its easy to compile and run it and see that it prints out three times ….

One next writes the “wrapper” java class - I’ve called it TestJavaTime.java - it only calls the time library with argument 11 - but this can easily be modified obviously

++

```
public class TestJavaTime {
   static {
        System.loadLibrary("ctime"); // myjni.dll (Windows) or libmyjni.so (Unix
es)
   }

   // Declare a native method average() that receives two ints and return a dou
ble containing the average
   private native long ctaitime(int clock);

   // Test Driver
   public static void main(String args[]) {
        System.out.println("TAI time via C called from Java is "  + new TestJava
Time().ctaitime(11));
   }
}
```

+++

Note the bits in red are explained below :

1) ctime the shared libary containing the c-function to be called is going to be called libctime.so
2) ctaitime(int clock) is the name of the C function this java class will be calling -

After writing this Java class we do :

```
> javac TestJavaTime.java
```

and then generate a header

```
> javah TestJavaTime
```

From the previous step a header file will have been generated - it must not be touched anymore - here is what it looks like - its name is TestJavaTime.h

++

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class TestJavaTime */

#ifndef _Included_TestJavaTime
#define _Included_TestJavaTime
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     TestJavaTime
 * Method:    ctaitime
 * Signature: (I)J
 */
JNIEXPORT jlong JNICALL Java_TestJavaTime_ctaitime
  (JNIEnv *, jobject, jint);

#ifdef __cplusplus
}
#endif
#endif
```

+++

This header has to be included in the function extracted from the first c program that I showed in this write-up. Note that the red bit in the header tells us what it neddsa to be called and what it's signature is :

```
JNIEXPORT jlong JNICALL Java_TestJavaTime_ctaitime (JNIEnv *, jobject, jint);
```

So here we go - here is the C function (note the header included -red) and the signature of the function (also red) - these

++

```
#include <jni.h>
#include <stdio.h>
#include "TestJavaTime.h"
#include <time.h>

JNIEXPORT jlong JNICALL Java_TestJavaTime_ctaitime
(JNIEnv *env, jobject thisObj, jint clock ) {
  long result;
  printf("Integer argument given to me is %d \n", clock);

  struct timespec gettime_now;
  clock_gettime(clock, &gettime_now);

  result = gettime_now.tv_nsec;

  if(clock == 11)  {// tai time)
    printf("I am the C code I will give Java the TAI time of  %ld \n", result);
  }
    else if (clock == 0) {
      printf("I am the C code I will give Java the REAL time of  %ld \n", result);
    }
    else if (clock == 1) {
      printf("I am the C code I will give Java the monotonic time of  %ld \n", result);
    }
    else {
            printf("Valid choices are only 1 (monotonic), 11 (TAI), 0 (UTC) \n");
    }
  return result;
}
```

++

Ok finally we need to give it a go - this will only compile and link (I found) with the flags shown - among other things the flags specify the location of the jni.h and jni_ md.h (a dependency of the jni.h)

Thus to build the shared library :

```
 > gcc -o libctime.so -fPIC -nostartfiles -shared -I
/usr/lib/jvm/java-8-openjdk-amd64/include  -I
/usr/lib/jvm/java-8-openjdk-amd64/include/linux TestJavaTime.c -lc
```

The java can be run but the library path has to be specified - annoyingly - and so here we go

```
>java -Djava.library.path=. TestJavaTime
Integer argument given to me is 11
I am the C code I will give Java the TAI time of  72873483
TAI time via C called from Java is 72873483
```

Here the printout is the printout from running the java - the time from inside the C function and from the java call are both printed.

```
> java -Djava.library.path=. TestJavaTime
Integer argument given to me is 11
I am the C code I will give Java the TAI time of  72873483
TAI time via C called from Java is 72873483
```

<Moving toward the CCS Version>

Next moving toward getting this integrated into CCS my first step was to create the class with main and C++ native function declaration(TimeGet.java) and then a class to store the times (TimeStorage.java) and a C++ function (timeaccess.C) that gets three times TAI time, UTC time. Using JNI once again the C++ function is called with an instance of TimeStorage.java - and the C++ code simply sets the values of the private data members of TimeStorage.java - the way to generate the JNI interface etc is as described in the previous example - for now TimeGet.java simply prints out the UTC, TAI and monotonic time in seconds and then nano-seconds we need to define what the interface to the user needs to be.

This time the C++ shared library is modifying the private data members of a Java Class - and then the time is stored in there.

The java and C++ classes/functions follow on the next 3 pages - these are trivial - only “subtlety” is accessing a Java class from C++ and modifying its data members.

<TimeGet.java> Only prints out times - need to think of a useful interface.

```
************************
public class TimeGet {

        static {
   System.loadLibrary("timeaccess"); // Load native library at runtime
   // hello.dll (Windows) or libtimeaccess.so (Unixes)
        }

        // Declare a native method timeaccess that receives the java storage class
TimeStorage and returns void
        private native void timeaccess(TimeStorage tstore);

        // Test Driver
        public static void main(String[] args) {
   TimeStorage ts = new TimeStorage();

   new TimeGet().timeaccess(ts);  // invoke the native method

   // print stuff out - from the filled TimeStorage class -
   System.out.println("TAI time (seconds): " + ts.getTimeSecsTAI());
   System.out.println("TAI time (nano-seconds): " + ts.getTimeNanoTAI());
   System.out.println("UTC time (seconds) : " + ts.getTimeSecsUTC());
   System.out.println("UTC time (nano-seconds) : " + ts.getTimeNanoUTC());
   System.out.println("MONOT time (seconds) : " + ts.getTimeSecsMONOT());
   System.out.println("MONOT (nano-seconds) : " + ts.getTimeNanoMONOT());

   }
}
********************************
```

<TimeStorage.java> Regular Java class stores TAI, UTC and monotonic times which have to be filled in by C or C++ code

****

```
//************************************
// stores TAI, UTC and MONOTONIC time in seconds and nano seconds -
// Author - Farrukh Azfar.
//********************************************

public class TimeStorage {

        // private data members storing seconds and nano-seconds -
        // TAI
        private long timeSecsTAI;
        private long timeNanoTAI;

        // UTC
        private long timeSecsUTC;
        private long timeNanoUTC;
        // Monotonic
        private long timeSecsMONOT;
        private long timeNanoMONOT;

        // set functions
        // TAI
        public void setTimeSecsTAI(long jjt) {
            timeSecsTAI = jjt;
        }
        public void setTimeNanoTAI(long kkt) {
             timeNanoTAI = kkt;
        }
        // UTC
        public void setTimeSecsUTC(long jju)
                timeSecsUTC = jju;
        }
        public void setTimeNanoUTC(long kku) {
           timeNanoUTC = kku;
        }
        // Monotonic
     public void setTimeSecsMONOT(long jjm) {
            timeSecsMONOT = jjm;
        }
        public void setTimeNanoMONOT(long kkm) {
            timeNanoMONOT = kkm;
        }


  // get methods
        // TAI
        public long getTimeSecsTAI() {
           return timeSecsTAI;
        }
        public long getTimeNanoTAI() {
           return timeNanoTAI;
        }

        // UTC
        public long getTimeSecsUTC() {
            return timeSecsUTC;
        }
        public long getTimeNanoUTC() {
            return timeNanoUTC;
        }

        // Monotonic
        public long getTimeSecsMONOT() {
            return timeSecsMONOT;
        }
        public long getTimeNanoMONOT() {
            return timeNanoMONOT;
        }
```

****

<TimeGet.h> - jni generated header for timeaccess.C

****

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class TimeGet */

#ifndef _Included_TimeGet
#define _Included_TimeGet
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:       TimeGet
 * Method:      timeaccess
 * Signature: (LTimeStorage;)V
 */
JNIEXPORT void JNICALL Java_TimeGet_timeaccess
  (JNIEnv *, jobject, jobject);

#ifdef __cplusplus
}
#endif
#endif
```

****

<timeaccess.C> C++ code to actual get the TAI UTC and monotonic times in seconds, nano seconds and then fill the TimeStorage Class passed to it by TimeGet.

****

```
//***********************************************
//JNI function to set data members Java class
//that carries the time information
// Author Farrukh Azfar
//***********************************************
#include <jni.h>
#include <stdio.h>
#include "TimeGet.h"
#include <time.h>

JNIEXPORT void JNICALL Java_TimeGet_timeaccess
(JNIEnv *env, jobject thisObj, jobject ts) {

  struct timespec gettime_utc; // utc
  struct timespec gettime_tai ; // tai
  struct timespec gettime_monot;//

  // fill in the structs ... 0, 11, 1 specify which time to fill the structs with -
  clock_gettime(0, &gettime_utc);
  clock_gettime(11, &gettime_tai);
  clock_gettime(1, &gettime_monot);

  // Set the values of all the functions in the class - here
  // get the pointer to the TimeStorage Class -

jclass cl = env->GetObjectClass(ts);

// find the ID of the field that the set member functions set -
 jfieldID fids_tai = env->GetFieldID(cl, "timeSecsTAI", "J");
 jfieldID fidns_tai = env->GetFieldID(cl, "timeNanoTAI", "J");
 jfieldID fids_utc = env->GetFieldID(cl, "timeSecsUTC", "J");
 jfieldID fidns_utc = env->GetFieldID(cl, "timeNanoUTC", "J");
 jfieldID fids_monot = env->GetFieldID(cl, "timeSecsMONOT", "J");
 jfieldID fidns_monot = env->GetFieldID(cl, "timeNanoMONOT", "J");
 // now set the field (equivalent to using the set member functions)...
  env->SetLongField(ts, fids_tai, gettime_tai.tv_sec);
  env->SetLongField(ts, fidns_tai, gettime_tai.tv_nsec);
  env->SetLongField(ts, fids_utc, gettime_utc.tv_sec);
  env->SetLongField(ts, fidns_utc, gettime_utc.tv_nsec);
  env->SetLongField(ts, fids_monot, gettime_monot.tv_sec);
  env->SetLongField(ts, fidns_monot, gettime_monot.tv_nsec);

}
*********************************************
```

Now create the shared library:

```
 gcc -o libtimeaccess.so -fPIC -nostartfiles -shared -I
/usr/lib/jvm/java-8-openjdk-amd64/include  -I
/usr/lib/jvm/java-8-openjdk-amd64/include/linux timeaccess.C -lc
```

...and run (you do need to specify java.library.path)

```
 java -Djava.library.path=. TimeGet
```