IBM

developerWorks®

Take a beginner's tour of the Informix virtual table interface with shared libraries

Write extensions to access external data sources and applications

Daniel Hebert
Carla Wilcox

March 17, 2011

IBM® Informix® provides access to external data sources through the virtual table interface (VTI). The VTI provides a set of hooks called *purpose functions*. As a developer, your task is to create an access method that implements VTI purpose functions and as many additional user-defined routines (UDRs) as necessary to access your external data source. This tutorial shows you how to compile and run your VTI UDR as a shared library.

Before you start

About this tutorial

Informix provides access to external data sources through the virtual table interface (VTI). External data sources include flat files, other databases, internet feeds, and other sources.

The VTI provides a set of hooks called *purpose functions*. As a developer, your task is to create an *access method* that implements VTI purpose functions and as many additional user-defined routines (UDRs) as necessary to access your external data source.

This tutorial shows you how to compile and run your VTI UDR as a shared library. For clarity in the examples, the VTI purpose functions reside in one source file and the auxiliary C functions reside in a separate source file. The VTI purpose functions are compiled and linked into a datablade library called *tutorial.bld*. The auxiliary C functions are compiled and linked into a separate shared library called *libtutorial*. One advantage of compiling the VTI purpose functions and the auxiliary C functions into separate libraries is that the auxiliary C functions can be used by multiple VTI libraries.

Prerequisites

You must have the Informix server installed. This tutorial was validated with Informix versions 11.50 and 11.70. Later versions are expected to behave similarly.

You need intermediate programming skills, as well as some background with the C programming language. You need a working knowledge of how to install and configure Informix. The Informix virtual appliance is used in this tutorial to provide a common environment. You can download the Informix virtual appliance from the IBM Informix free downloads site, which is listed in the Related topics section. You can also find the Informix virtual appliance by searching for IBM virtual appliances on the IBM web site. Or you can go to the IBM free downloads web page and search for **Informix virtual appliance**.

You can find documentation for the Informix virtual appliance in the Informix Information Center and on the appliance itself once the image is booted.

System requirements

This tutorial assumes you are using Linux, but the basic concepts apply and the demo code can be ported to any system that supports Informix.

Overview

This tutorial demonstrates how to use a shared library in conjunction with the VTI. Shared libraries have the advantage that they are loaded dynamically and only when necessary. This limits your process to grow only when needed.

This tutorial also shows you how to debug UDR code that resides in a shared library. Note that the symbols in the shared library are not visible until the shared library is loaded.

The shared library in this demonstration implements a print method. This method is used to implement a trace facility for the VTI purpose functions invoked during a simple select SQL statement. You can build on this demonstration to create your own custom C library. Your customized library is available for use by VTI purpose functions and can perform any general task.

This demonstration VTI UDR shows only four of the complete set of purpose functions. The server calls a purpose function to perform well-defined tasks to access and manage your external data sources. You will need to implement all of the remaining purpose functions for a complete VTI UDR. Refer to Related topics for larger, more advanced examples.

Understanding and compiling the demo

Directory Structure

Start by downloading and unzipping the tutorial.zip file from the download section of this tutorial. The tutorial code is organized under the relative path `tutorial`. The zipped files and their descriptions are as follows:

tutorial/tut_shared/sample.c
  C source code that provides auxiliary support functions used by the *tutorial* access method purpose functions.

tutorial/tut_shared/sample.h
  C source code header file for the auxiliary support functions used by the *tutorial* access method purpose functions. Included by C sources that make calls to any of these functions.

tutorial/tut_shared/makefile
: Compiles the auxiliary support functions into the shared library libtutorial.so.

tutorial/tutorial.c
: C source code that provides the *tutorial* access method purpose functions. Additional commonly used headers have been added for your development convenience.

tutorial/makefile
: Compiles the *tutorial* access method purpose functions into the library tutorial.bld.

tutorial/tutorial.sql
: SQL file that creates the *tutorial* access method and defines its purpose functions.

tutorial/tutorial_example.sql
: SQL file that creates an example table using the *tutorial* access method.

tutorial/drop_tutorial.sql
: SQL file that drops the *tutorial* access method and its associated purpose functions.

tutorial_drop_tutorial_example.sql
: SQL file that drops the *example* table.

tutorial/space.sh
: Example shell script that creates an sbspace. Note that the demo_on database on the Informix virtual appliance already defines an sbspace, so it is not necessary to explicitly create an sbspace.

Compiling the UDR code

This tutorial has been tested on the 11.70 Informix virtual appliance. If you are running on another operating system, you might need to port the code. The following steps describe how to compile the tutorial on the 11.70 Informix virtual appliance.

1. Compile the tutorial as user `root`. Log in as root and unpack the tutorial code so that it resides in `$INFORMIXDIR/extend/tutorial`. In this example, $INFORMIXDIR is assumed to be `/opt/IBM/informix`. Modify the instructions appropriately if your INFORMIXDIR is something other than `/opt/IBM/informix`.
2. Compile the sources in `/root/tutorial/tut_shared` as shown in Listing 1.

   ### Listing 1. Compiling sources

   ```
   cd /opt/IBM/informix/extend/tutorial/tut_shared
   make
   ```
3. Clean any previous builds, as shown in Listing 2.

   ### Listing 2. Cleaning previous builds

   ```
   rm -f sample
   rm -f *.o
   rm -f *.so*
   rm -f /opt/IBM/informix/lib/libtutorial.so.1
   ```
4. Compile the auxiliary code into libtutorial.so*, as shown in Listing 3.

   ### Listing 3. Compiling the auxiliary code

   ```
   gcc -Wall -fPIC -c *.c -I.
   gcc -shared -Wl,-soname,libtutorial.so.1 -o libtutorial.so.1.0 *.o
   ```

5. Change the ownership on the files, and create several links that set up various library versions, as shown in Listing 4. The additional links allow for proper execution on other UNIX-like operating systems. Changing the ownership of the files is not strictly necessary, but it is a best practice for user *informix* to own files in the `$INFORMIXDIR/extend` tree.

   ### Listing 4. Changing ownership

   ```
   chown informix:informix .;
       su informix -c "ln -s libtutorial.so.1.0 libtutorial.so";
       su informix -c "ln -s libtutorial.so.1.0 libtutorial.so.1"
   chown informix:informix *
   ```
6. Copy the shared library to `$INFORMIXDIR/lib`, and set its owner to *informix*, as shown in Listing 5.

   ### Listing 5. Setting owner to informix

   ```
   cp libtutorial.so.1.0 /opt/IBM/informix/lib/libtutorial.so.1;
       chown informix:informix /opt/IBM/informix/lib/libtutorial.so.1
   ```
7. Compile the sources in `/opt/IBM/informix/extend/tutorial`, as shown in Listing 6.

   ### Listing 6. Compiling sources

   ```
   cd /opt/IBM/informix/extend/tutorial
   make
   ```
8. Clean any previous builds, as shown in Listing 7.

   ### Listing 7. Cleaning previous builds

   ```
   rm -f *.bld *.o *.so* *~
   rm -f /usr/lib/libtutorial.so.1
   ```
9. Change the ownership of the current directory by entering **chown informix:informix** .
10. Compile the access method purpose functions, as shown in Listing 8.

    ### Listing 8. Compiling access method purpose functions

    ```
    gcc -g -fPIC -I/opt/IBM/informix/incl/public
        -I/opt/IBM/informix/extend/tutorial/tut_shared
        -L/opt/IBM/informix/extend/tutorial/tut_shared -c tutorial.c
    gcc -shared -g -fPIC -I/opt/IBM/informix/incl/public
        -I/opt/IBM/informix/extend/tutorial/tut_shared
        -L/opt/IBM/informix/extend/tutorial/tut_shared
        -o tutorial.bld tutorial.o -ltutorial
    ```
11. Create several links that set up various library versions, as shown in Listing 9. The additional links allow for proper execution on other UNIX-like operating systems.

    ### Listing 9. Setting up library versions

    ```
    su informix -c "ln -s
      /opt/IBM/informix/extend/tutorial/tut_shared/libtutorial.so.1.0 libtutorial.so"
    su informix -c "ln -s
      /opt/IBM/informix/extend/tutorial/tut_shared/libtutorial.so.1.0 libturorial.so.1"
    ```
12. Create a link to the tutorial shared library from /usr/lib, as shown in Listing 10. This tutorial assumes that /usr/lib is automatically searched and that it need not be included in the LD_LIBRARY_PATH environment variable. If this is not the case for your environment, add **$INFORMIXDIR/extend/tutorial/tut_shared**t to LD_LIBRARY_PATH.

Listing 10. Creating link to shared library

```
ln -s /opt/IBM/informix/extend/tutorial/tut_shared/libtutorial.
  so.1.0 /usr/lib/libtutorial.so.1
```

13. Change the ownership of the files, as shown in Listing 11. This is not strictly necessary, but it is a best practice for user *informix* to own files in the `$INFORMIXDIR/extend` tree.

Listing 11. Changing ownership

```
chown informix:informix *
```

14. Bring the Informix server down by typing **onmode -ky** to configure the server.
15. Modify the server onconfig file to define a VPCLASS for the UDR code that executes when the *tutorial* access method purpose functions are called, as shown in Listing 12. This is a best practice. Executing your UDR code in its own VP isolates your UDR code from the Informix server and therefore reduces the risk of your UDR code which causes the Informix server to crash or abort. This makefile first removes any prior definitions for a tutorial VPCLASS and then adds the VPCLASS. The new VPCLASS does not suffer from a decline in performance over time due to priority aging by the operating system.

Listing 12. Modifying the onconfig file

```
awk 'BEGIN { } { if ( ! /VPCLASS tutorial,num=1,noage/ ) print; }'
    /opt/IBM/informix/etc/onconfig.demo_on >/tmp/onconfig
cp tmp/onconfig /opt/IBM/informix/etc/onconfig.demo_on
echo "VPCLASS tutorial,num=1,noage"
    >>/opt/IBM/informix/etc/onconfig.demo_on
```

16. Set the permissions on the library that contains the *tutorial* access method purpose functions, as shown in Listing 13.

Listing 13. Setting permissions

```
chmod 555 /opt/IBM/informix/extend/tutorial/tutorial.bld
```

17. Optionally echo the value of LD_LIBRARY_PATH for informational purposes, as shown in Listing 14.

Listing 14. Echoing LD_LIBRARY_PATH

```
LD_LIBRARY_PATH = /opt/IBM/informix/clidriver/lib:
    /opt/IBM/informix/lib/esql:
    /opt/IBM/informix/lib/dmi:
    /opt/IBM/informix/lib/cli:
    /opt/IBM/informix/lib/c++:
    /opt/IBM/informix/lib
```

18. Start the Informix server by typing **oninit**.
19. Once the server is up, use the **onstat -g sch** command to verify that your VP is correctly defined. The tutorial VP should be listed in both the Scheduler and Thread Migration Statistics sections.

Configuring the sbspace and defining the tutorial access method and UDRs

Configuring the sbspace

If you are using the demo_on Informix instance on Informix virtual appliance, an sbspace is already defined and available for your use. In this case, you don't need to do anything further.

If you need to define an sbspace, modify the shell script space.sh if necessary and then execute it. This shell script creates a chunk and then adds the sbspace. Type the **onstat -d** command to display your chunks and spaces. You should see your sbspace listed.

Defining the tutorial access method and UDRs

At this point in the tutorial, you have compiled the UDR code into two libraries: one library that contains the *tutorial* access method purpose functions and one library that contains the auxiliary function used by the purpose functions. You have also configured the Informix server so that it can support the new access method.

The next step is to define the access method and its functions so that the Informix server can use them. The following steps describe how to execute the SQL code. The file tutorial/tutorial.sql file creates the tutorial database and defines the UDR routines and the *tutorial* access method. The file tutorial/drop_tutorial.sql undefines the UDR routines and the *tutorial* access method and then drops the tutorial database.

1. Log in as user **informix**.
2. Execute the tutorial.sql file by entering the code in Listing 15.

   #### Listing 15. Executing tutorial.sql

   ```
   cd /opt/IBM/informix/extend/tutorial
   dbaccess - tutorial.sql
   ```

   The first SQL statements create the tutorial database, as shown in Listing 16.

   #### Listing 16. Creating the tutorial database

   ```
   create database tutorial;
   database tutorial;
   ```

   The next SQL statements define each of the UDR routines, as shown in Listing 17.

   #### Listing 17. Defining the UDR routines

   ```
   create function tutorial_open (pointer)
   returns integer
   with (variant, class="tutorial")
   external name
   "$INFORMIXDIR/extend/tutorial/tutorial.bld(tutorial_open)"
   language c;
   grant execute on function tutorial_open (pointer) to public;
   ```

Note the following about the tutorial_loadlib function: This function is not a proper access method purpose function. It is included with the access method purpose functions as a best practice. The Informix server calls this function when it executes the SQL statement `execute function tutorial_loadlib()`. Providing this function and executing this SQL statement ensures that the library can be loaded and that the UDR is correctly invoked and executed. After the UDR routes are defined, the *tutorial* access method is defined. This definition implements that function. For example, `am_beginscan` maps to the UDR, as shown in Listing 18.

Listing 18. Mapping code to the UDR

```
tutorial_beginscan.
create primary access_method tutorial (
    am_beginscan = tutorial_beginscan,
    ...
    am_update = tutorial_update );
```

3. If you want to undo the work that tutorial.sql did, you can execute drop_tutorial.sql, as shown in Listing 19.

Listing 19. Undoing tutorial.sql

```
cd /opt/IBM/informix/extend/tutorial
dbaccess - drop_tutorial.sql
```

Note that you should drop the *tutorial* access method before you drop the UDR routines. An error message results if you drop the UDR routines first.

Execute the tutorial access method purpose functions

At this point in the tutorial, you have defined the tutorial access method and its UDR routines. Complete the following steps to issue SQL statements that cause the Informix server to invoke the VTI UDR routines.

1. Execute the SQL in tutorial_example.sql by entering the code in Listing 20.

Listing 20. Executing the SQL

```
cd /opt/IBM/informix/extend/tutorial
dbaccess - tutorial_example.sql
```

This SQL file loads the tutorial UDR library and then creates a table called *example* that resides in the sbspace and uses the tutorial access method. When the Informix server executes these SQL statements, it invokes tutorial access purpose functions to augment or complete the requested tasks. In this tutorial's demonstration, these methods simply call the auxiliary function `print()` that resides in the tutorial shared library. The Informix server's online log file contains the output. Use the command **onstat -m** to display the last part of your Informix server online log file, as shown in Listing 21.

Listing 21. Viewing the online log file

```
trace: loadlib
trace: create
trace: open
trace: close
```

2. If you want to undo the work done by tutorial_example.sql, enter the drop_tutorial_example.sql commands, as shown in Listing 22.

Listing 22. Dropping the tutorial example

```
cd /opt/IBM/informix/extend/tutorial
dbaccess - drop_tutorial_example.sql
```

Debugging

Debugging with gdb, multiple shared libraries, and your UDRs

When shared libraries are part of an application, debugging can be challenging. One of the most common challenges is gaining access to symbols in your shared libraries. Before the symbols in your shared library are available to gdb, the shared library must first be loaded.

Following are some tips to make debugging with shared libraries a bit easier:

- Make sure that all the libraries are compiled with the compiler debug flag, such as **-g** if you are using cc or gcc.
- Use the gdb **dir** and **source** commands to specifically point to source code for your shared libraries.
- Use the dl interface, such as dlopen, dlsym, dlclose, and so on, for explicit control over loading your shared libraries. To do this, add **-ldl** to your compilation commands. This adds a greater amount of control over what gets loaded at the expense of added complexity and potential platform-specific issues. Use this approach as a last resort.
- Be generous with breakpoints. If you find that you are dealing with threads, a long line of shared libraries, or gdb hanging on methods that should be returning, you can set breakpoints on the lines afterward so that the program continues using the `c` command in gdb.

The following steps offer one approach to debugging your UDRs.

1. Let your Informix server run as usual.
2. Make sure that your shared library is loaded. For example, to make sure that the tutorial.bld library is loaded, run **dbaccess** and manually execute the SQL statement **execute function tutorial_loadlib();**.
3. Find the process ID for the tutorial VP by issuing the command **onstat -g glo** command. You should see output that looks similar to Listing 23. In this example, the PID for your tutorial VP is `11411`.

Listing 23. Output to find the PID

```
IBM Informix Dynamic Server Version 11.70.UC1DE -- On-Line -- Up 00:01:06 -- 177036 Kbytes
```

```
MT global info:
sessions threads  vps      lngspins
0        27       14       0

          sched calls     thread switches yield 0   yield n   yield forever
total:    56749           2320            489       710       657
per sec:  0               0               0         0         0

Virtual processor summary:
 class       vps       usercpu   syscpu    total
 cpu         1         0.11      0.49      0.60
 aio         3         0.00      0.02      0.02
 lio         1         0.00      0.00      0.00
 pio         1         0.00      0.00      0.00
 adm         1         0.00      0.04      0.04
 soc         1         0.00      0.01      0.01
 msc         1         0.00      0.00      0.00
 jvp         1         0.00      0.00      0.00
 fifo        1         0.00      0.00      0.00
 bts         1         0.00      0.00      0.00
 idsxmlvp    1         0.00      0.00      0.00
 tutorial    1         0.04      0.02      0.06
 total       14        0.15      0.58      0.73

Individual virtual processors:
 vp    pid       class       usercpu   syscpu    total     Thread    Eff
 1     11406     cpu         0.11      0.49      0.60      1.98      30%
 2     11407     adm         0.00      0.04      0.04      0.00       0%
 3     11408     bts         0.00      0.00      0.00      0.00       0%
 4     11409     idsxmlvp    0.00      0.00      0.00      0.00       0%
 5     11410     jvp         0.00      0.00      0.00      0.00       0%
 6     11411     tutorial    0.04      0.02      0.06      0.29      20%
 7     11412     lio         0.00      0.00      0.00      0.02       0%
 8     11413     pio         0.00      0.00      0.00      0.01       0%
 9     11414     aio         0.00      0.02      0.02      0.08      25%
 10    11415     msc         0.00      0.00      0.00      0.01       0%
 11    11416     fifo        0.00      0.00      0.00      0.01       0%
 12    11417     aio         0.00      0.00      0.00      0.01       0%
 13    11418     soc         0.00      0.01      0.01      NA         NA
 14    11419     aio         0.00      0.00      0.00      0.01       0%
                 tot         0.15      0.58      0.73
```

3. Run gdb as user root, and attach to your tutorial VP **gdb -pid 11411**. When gdb comes up, you should be able to set breakpoints in your VTI UDRs, as shown in Listing 24.

Listing 24. Setting breakpoints

```
gdb --pid 11411
GNU gdb (GDB; SUSE Linux Enterprise 11) 6.8.50.20081120-cvs
Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i586-suse-linux".
For bug reporting instructions, please see:
<http://bugs.opensuse.org/>.
Attaching to process 11411
Reading symbols from /opt/IBM/informix/bin/oninit...(no debugging symbols found)...done.
Reading symbols from /lib/libpthread.so.0...(no debugging symbols found)...done.
...
Reading symbols from /opt/IBM/informix/extend/tutorial/tutorial.bld...done.
Loaded symbols for /opt/IBM/informix/extend/tutorial/tutorial.bld
Reading symbols from /usr/lib/libtutorial.so.1...done.
```

```
Loaded symbols for /usr/lib/libtutorial.so.1
0xffffe430 in __kernel_vsyscall ()
(gdb) b tutorial_loadlib
Breakpoint 1 at 0xb788c52e: file tutorial.c, line 36.
(gdb) c
Continuing.
```

4. In another window, run **dbaccess**.
5. Execute an SQL statement that causes the Informix server to invoke your tutorial_loadlib() UDR, such as **dbaccess - tutorial_example.sql**.
6. See the breakpoint you set above, as shown in Listing 25.

Listing 25. Breakpoint results

```
Breakpoint 1, tutorial_loadlib (fparam=0x4c842920) at tutorial.c:36
36  print("loadlib");
(gdb) s
38  return MI_TRUE;
(gdb) c
Continuing.
```

Conclusion

The virtual table interface (VTI) is a powerful Informix Dynamic Server feature that enables users to write extensions to access external data sources and applications. This tutorial showed how to compile and link shared libraries with VTI applications as well as how to write an access method and its purpose functions. This tutorial reviewed using purpose functions and debugging a VTI application using GDB. With this general knowledge, you can begin developing customized applications using the virtual table interface.

Downloadable resources

Description	Name	Size
Sample code for this tutorial	tutorial.zip	5KB

Related topics

- Go to the IBM Informix free product download site to download the Informix virtual appliance.
- See the Informix Information Center to learn more about Informix virtual tables.
- Link to the downloadable publications for the IBM Informix 11.7 family of products, especially dapif.pdf, dapip.pdf, ids_udr_bookmap.pdf, vii.pdf, and vti.pdf.
- Look up the Informix area on developerWorks to get the resources you need to advance your Informix skills.
- Download an evaluation version or a free edition of Informix 11.7.