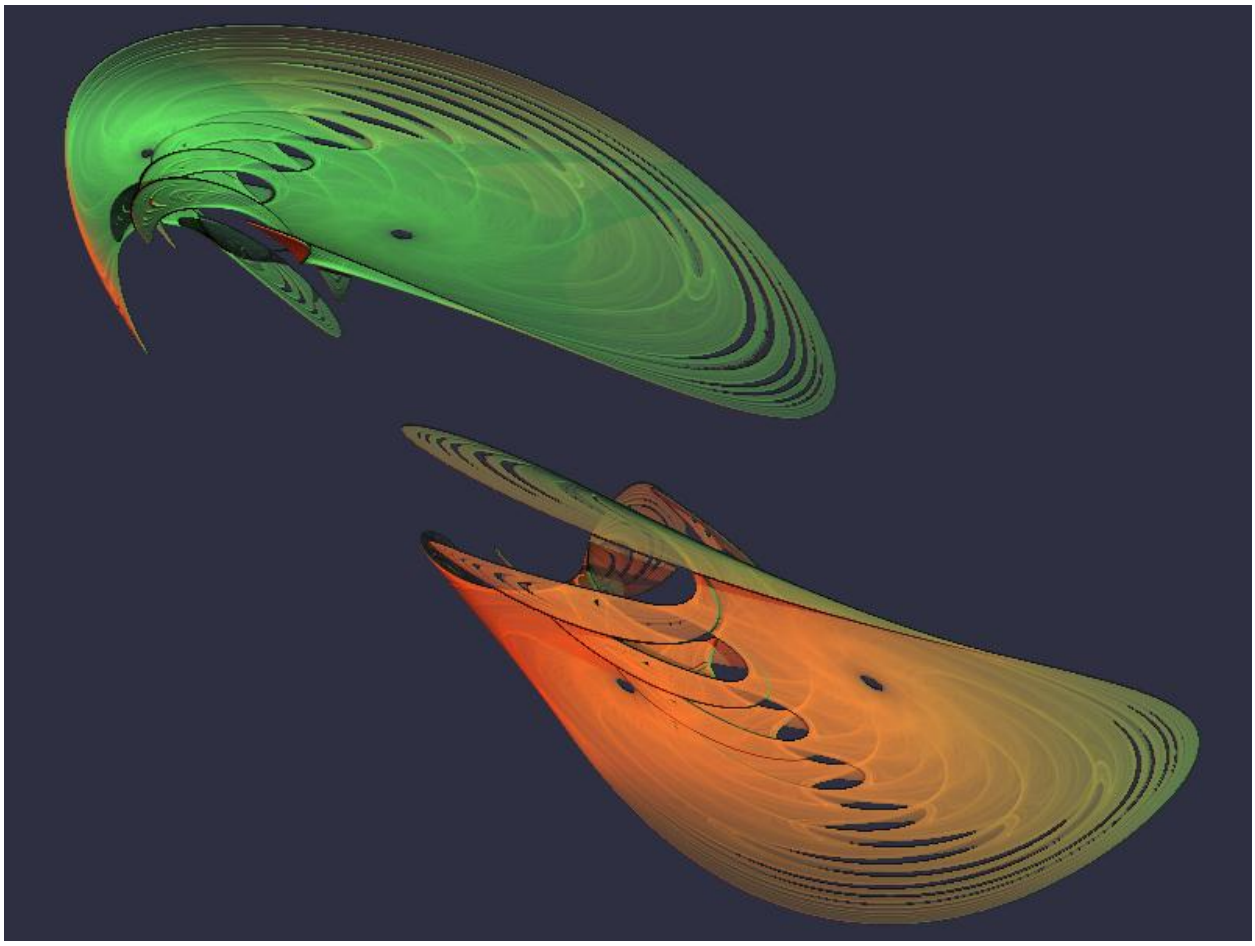
DX Debugging Tools

Using DXTell 1.0



Author: Ian Tree
Owner: HMNL b.*v.*
Customer: Public
Status: Final
Date: 02/04/2012 12:27
Version: 1.0.0

Disposition: Open Source

Document History

Document Usage

This is an open source document you may copy and use the document or portions of the document for any purpose.

Revision History

Date of this revision: 02/04/2012 12:27	Date of next revision *None*

Revision Number	Revision Date	Summary of Changes	Changes marked
1.0.0	24/02/10	Initial Base Version	No
1.0.0	02/04/12	QE Version	No

Acknowlegements

Frontpiece Design was produced by the chaoscope application.

Contents

1. Introduction to DXTell

DXTell is a command processor that is used to allow Domino Server Add-Ins to be tested on a workstation by providing the ability to pass commands to the Add-Ins Message Queue (MQ).

Little Known Fact #247

All of the Server API calls and functionality that is required to support a Server Add-In task is also available in the Notes Workstation code.

The DXTell application is constructed as a Domino Command Line Processor, it can be run from a command window on a workstation.

DXTell is a C++ application constructed on the DXCommon application kernel supporting Domino version from 6.0 through 8.5 on both Widows Notes Workstations.

2. Using DXTell

When initially testing a new or updated Server Add-In Task the first thing that you need to do is to prepare a test server to do the testing on. While it is extremely easy to install a new Domino server it can be time consuming to prepare that environment just for initial "kick the tyres" testing. Also if your code causes a PANIC or other fatal condition you may have to wade your way through a rather large NSD to locate an obvious (when you have seen it) but trivial problem. Well, there is a better way.

Little Known Fact #247

All of the Server API calls and functionality that is required to support a Server Add-In task is also available in the Notes Workstation code.

2.1 Testing Methodology

Install the Server Add-In task in your Notes Executable directory. Have any local databases that are needed by your Add-In available on the client. Shut down your Notes Client, this minimises the volume of extraneous clutter that will appear in any NSD that you might have. Open two DOS Prompt windows. In the first window you launch your server Add-In by typing what you would type as parameters to the Load command if you were running your Add-In task on a server and your Add-In will start and, hopefully, begin it's normal processing. Should a fault occur then you will be offered the choice of debugging the problem with any of your installed debuggers. The second DOS prompt window that you opened is for the cunning bit. When running your Add-In on a server you control your Add-In task through the use of "Tell" command entered through the server console. From the second window you can use the DXTell program to pass commands to your Add-In in the same way that you would on a server.

e.g. DXTell MyAddin stop

This will pass a "stop" command to the Add-In task, assuming that it has created a Message Queue (MQ) with a name of "MyAddin" and that the Add-In task is correctly monitoring that Message Queue (MQ).

The second DOS Prompt Window can also be useful for running NSD commands. If you suspect that your program is looping you can use the "nsd -kill" or "nsd -dumpandkill" commands to terminate or terminate and dump your application. You can also use the "nsd -monitor" command to start an interactive NSD session, this can be particularly useful to inspect the stacks of your application at regular intervals to verify correct execution or to aid with problem diagnosis. Take note of the pid of your application, this will be reported when nsd attaches to it, then periodically issue the "DUMP 0x<your pid>" command (the pid is reported in hex so it is vital to prefix it with "0x").

When you have finished you should issue the "DETACH" command to detach nsd from all of the processes and then issue the "QUIT" command to shut nsd down.

3. Installing DXTell

3.1 Building the DXTell Command Processor

The DXTell command processor is built on the Domino eXplorer Tools DXCommon kernel and the Notes C API, you need to download and install these before you can build the DXTell application.

3.1.1 Reference Environments

DXTools and the DXCommon kernel are portable across multiple platforms that support the Notes API. However there are a limited set of reference environments on which they are regularly built and regression tested.

Windows:

Build Environment:

Microsoft Visual Studio 2005

Version 8.0.50727.867 (vsvista.050727-8600)

Running on any supported windows workstation.

Note: Backward compatibility tests are done with Visual Studio 2003 as that is the officially supported development platform for the Notes API.

Notes API Version 8.5.

Execution Environment:

Windows Standard Server 2008 R2 (32 bit).

Domino Server 8.5.1 FP3.

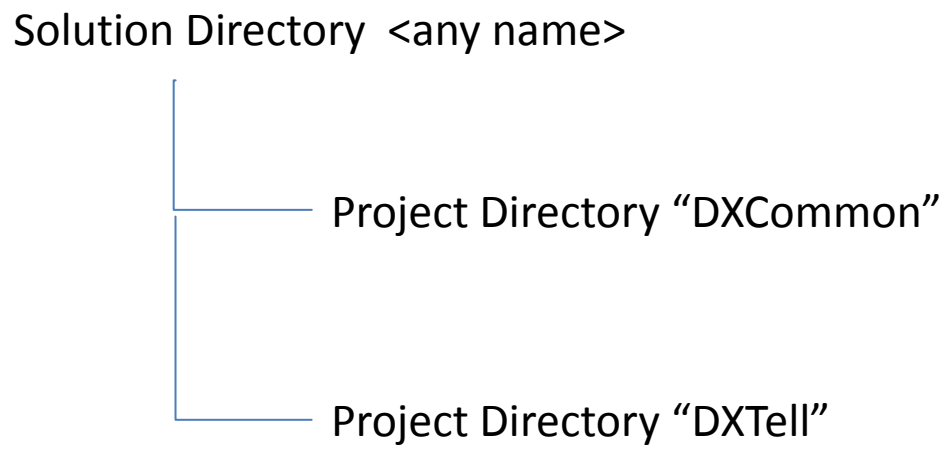
Note: Execution environments from Domino 6.5.x through 8.5.x are regularly used.

3.1.2 Notes API Installation

For both Windows and Linux DXTools assumes that the Notes API is installed in the default configuration specified in the API documentation.

3.1.3 Directory Structure

For both Windows and Linux DXTools uses a reference development directory structure based on the Visual Studio structure.



In Visual Studio the DXCommon project directory should have the “Do Not Build” property set.

In both the Windows and Linux environments it is possible to use a symbolic link for the ”DXCommon” directory. This is a common deployment pattern for development environments where different versions of an API might need to be supported.

3.1.4 Installing the DXCommon Kernel Sources

Windows:

The DXCommon kernel is supplied as a zipped archive (.zip). The contents of the archive should be unpacked to either the <solution directory>\DXCommon directory or unpacked to a directory that will then be used as the base for a symbolic link from the <solution directory>\DXCommon directory.

As an example.

Unpack the DXCommon kernel into a directory “c:\usr\include\DXCommon-3.12.0” and then create the symbolic link from within the solution directory using the following command.

mklink /D DXCommon “c:\usr\include\DXcommon-3.12.0”

3.1.5 Installing the DXTell Sources

Windows:

The DXTell sources are supplied as a zipped archive (.zip). Create an empty project called “DXTell” in the <solution directory>. Then unpack the contents of archive into the project directory and add each of the source and header files to the project.

Header Files

AppRunSettings.h

DXTell.h

Source Files

AppRunSettings.cpp

DXTell.cpp

3.1.6 Build Settings

Windows:

Add each source and header file that is used from the DXCommon kernel to the DXTell project. To populate the individual folder right-click on the filter then select “Add” then “Existing Item” navigate to the required source or header file(s), select the file(s) and click the “Add” button. The contents of each filter are listed below.

Header Files

DXCommon\APIPackages.h

DXCommon\DXException.h

DXCommon\DXGlobals.h

DXCommon\ElapsedTimer.h

DXCommon\ExecEnvironment.h

DXCommon\Debug\Helper.h

DXCommon\Platform\NotesBase.h

DXCommon\Platform\PlatBase.h

DXCommon\RunSettings.h

Source Files

DXCommon\APIPackages.cpp

DXCommon\DXException.cpp

DXCommon\ElapsedTimer.cpp

DXCommon\ExecEnvironment.cpp

DXCommon\Debug\Helper.cpp

DXCommon\RunSettings.cpp

The following non-default settings should then be made to the project settings. Any other settings should not prevent a successful build.

Section/Entry	Release Setting	Debug Setting
General		
Character Set	Not Set	Not Set
C/C++		
Preprocessor		
Preprocessor Definitions	WIN32;NDEBUG;_CONSOLE;W32	WIN32;_DEBUG;_CONSOLE;W32
Code Generation		
Runtime Library	Multi-threaded (/MT)	Multi-threaded Debug DLL (/MTd)
Struct Member Alignment	1 Byte (/Zp1)	1 Byte (/Zp1)
Command Line		
Additional Options	/Oy-	/Oy-
Linker		
Input		
Additional Dependencies	notes.lib	notes.lib Dbghelp.lib Psapi.lib

Notes:

Static linking of the runtime is used as since the advent of Side-By-Side (SXS) assembly of applications it is increasingly common to find server environments that do not have the latest C/C++ Runtime manifests installed.

/Zp1 packing is a Notes API requirement as all Notes API structures are packed and not padded or member aligned.

/Oy- is an important setting, without it the compiler will use the Frame Pointer as a general purpose register rather than pointing to the current frame, this will cause any NSD dump to be complete garbage and make debugging virtually impossible.

The additional libraries for the debug settings Dbghelp.lib and Psapi.lib are used to enable additional debug capabilities such as memory leak detection that are provided by DXCommon kernel modules.

3.1.7 Building and Deploying the Application

Windows:

Select “Build” and then “Build DXtell”.

Copy the resulting executable (DXTell.exe) and the associated Program Debug Database (DXTell.pdb) to the Notes Executable directory on the workstation where you want to run the Command Processor.

4. Running DXTell

When running your Add-In on a server you control your Add-In task through the use of "Tell" command entered through the server console. From the second window you can use the DXTell program to pass commands to your Add-In in the same way that you would on a server.

e.g. DXTell MyAddin stop

This will pass a "stop" command to the Add-In task, assuming that it has created a Message Queue (MQ) with a name of "MyAddin" and that the Add-In task is correctly monitoring that Message Queue (MQ).