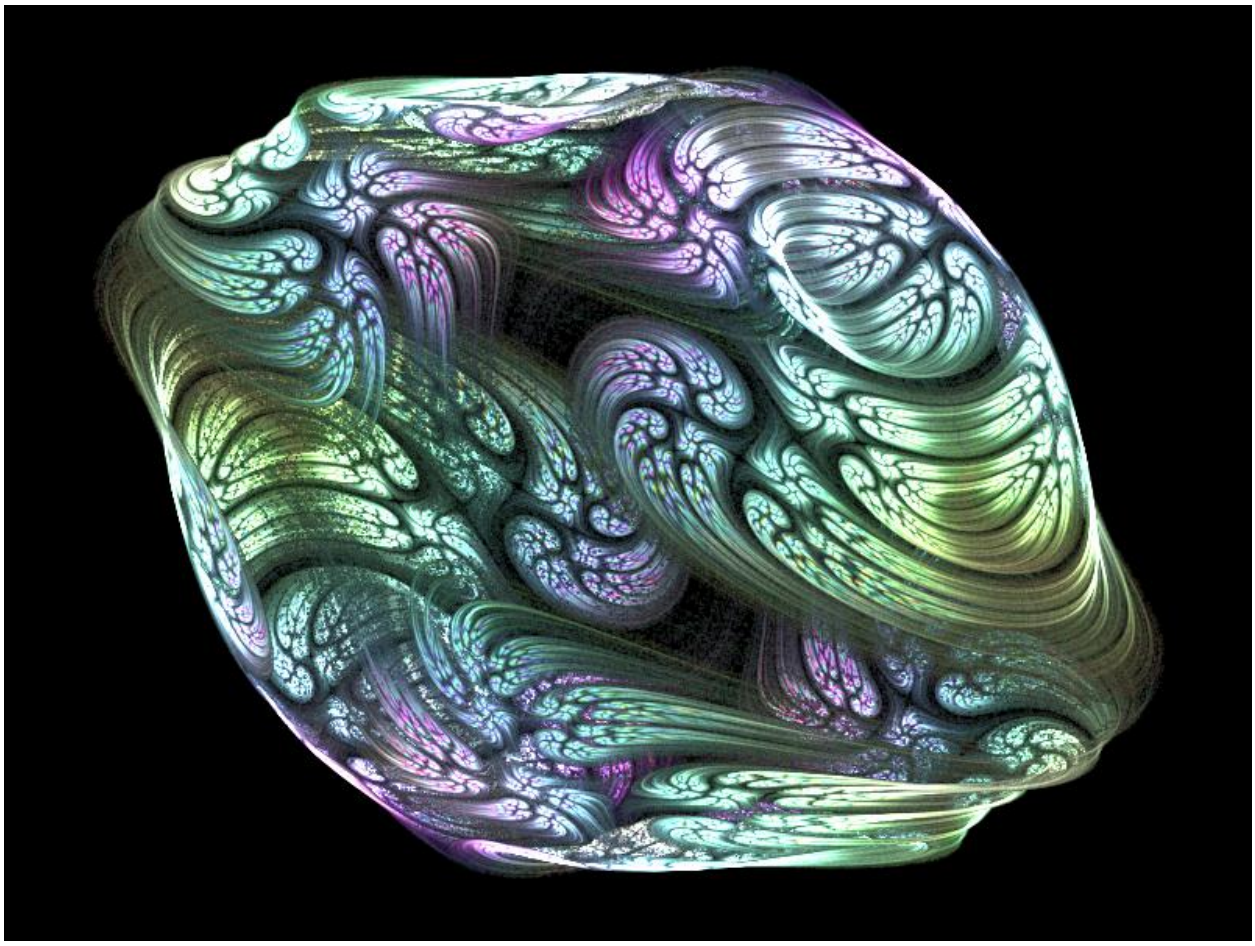
DX Debugging Tools

Using DXIPA 1.0



Author: Ian Tree
Owner: HMNL b.*v.*
Customer: Public
Status: Final
Date: 02/04/2012 11:10
Version: 1.0.1

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 11:10	Date of next revision *None*

Revision Number	Revision Date	Summary of Changes	Changes marked
1.0.0	17/10/11	Initial Base Version	No
1.0.1	02/04/12	QE Version	No

Acknowlegements

Frontpiece Design was produced by the chaoscope application.

IBM, the IBM Logo, Domino and Notes are registered trademarks of International Business Machines Corporation.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation.

Linux is a registered trademark of Linus Torvalds.

UNIX is a registered trademark of The Open Group.

Contents

1. Introduction to DXIPA

The DXIPA tool is provided to extract readable output from an Instrumentation Package Recording (.ipr) file that is produced by the DX Kernel when build in debug (_DEBUG) or Instrumented (DXIP) configuration. The application can list records in a recording stream, produce frequency tables in text or histogram form according to the command line options specified. All outputs from DXIPA are written to the standard output file (STDOUT).

2. Using DXIPA

The DXIPA tool is provided to extract readable output from an Instrumentation Package Recording (.ipr) file. The application can list records in a recording stream, produce frequency tables in text or histogram form according to the command line options specified. All outputs from DXIPA are written to the standard output file (STDOUT).

Syntax:

DXIPA <ipr file name> [-D][-S][-C]

The –D switch produces the output in list (dump) format, each record is annotated from the stream in the order that they appear. A short sample is shown below with an explanation of the output.

```
DX Instrumentation Package Analyser (DXIPA) Version: 1.0.0 build: 02 analysing:
/home/domino/cyclotron/IBM_TECHNICAL_SUPPORT/DXIP-QCopy20111229-124517.ipr
+0102584, TID=9, Evt=1, Prio=10, Dwell=0, Mill=126.
+0102576, TID=8, Evt=1, Prio=10, Dwell=0, Mill=230.
+0102580, TID=7, Evt=1, Prio=15, Dwell=0, Mill=442.
+0102710, TID=9, Evt=2, Prio=15, Dwell=21, Mill=366.
+0103022, TID=7, Evt=2, Prio=11, Dwell=21, Mill=90.
+0102806, TID=8, Evt=2, Prio=10, Dwell=23, Mill=350.
+0103097, TID=9, Evt=3, Prio=11, Dwell=21, Mill=141.
+0102598, TID=10, Evt=1, Prio=15, Dwell=0, Mill=660.
+0103133, TID=7, Evt=3, Prio=11, Dwell=21, Mill=151.
+0102604, TID=3, Evt=2, Prio=15, Dwell=21, Mill=702.
```

Each line in the output shows that values collected from a single instrumentation record. The first column **+0102584** shows the elapsed time in milliseconds since the application was started that the record was collected. The second column **TID=9** shows the identity of the worker thread from which the record was collected. The third column **Evt=1** shows the relative event number on the thread this allows sequencing of events from a single thread. The fourth column **Prio=10** shows the priority at which the request was dispatched to the worker thread. The fifth column **Dwell=0** shows the number of milliseconds that the worker thread was idle before processing the current request. The sixth column **Mill=126** shows the elapsed time in milliseconds that the current request took to complete execution.

The –S switch tells the analyser application to collate statistics on the stream in the recording file. A sample is shown below with explanation.

```
Mill Times.

 3250 - 3299: 1.
 3200 - 3249: 1.
 3150 - 3199: 0.
 3100 - 3149: 0.
 3050 - 3099: 1.
 3000 - 3049: 1.
 2950 - 2999: 2.
 2900 - 2949: 0.
 2850 - 2899: 5.
 2800 - 2849: 3.
 2750 - 2799: 0.
 2700 - 2749: 1.
 2650 - 2699: 2.
 2600 - 2649: 2.
 2550 - 2599: 3.
 2500 - 2549: 4.
```

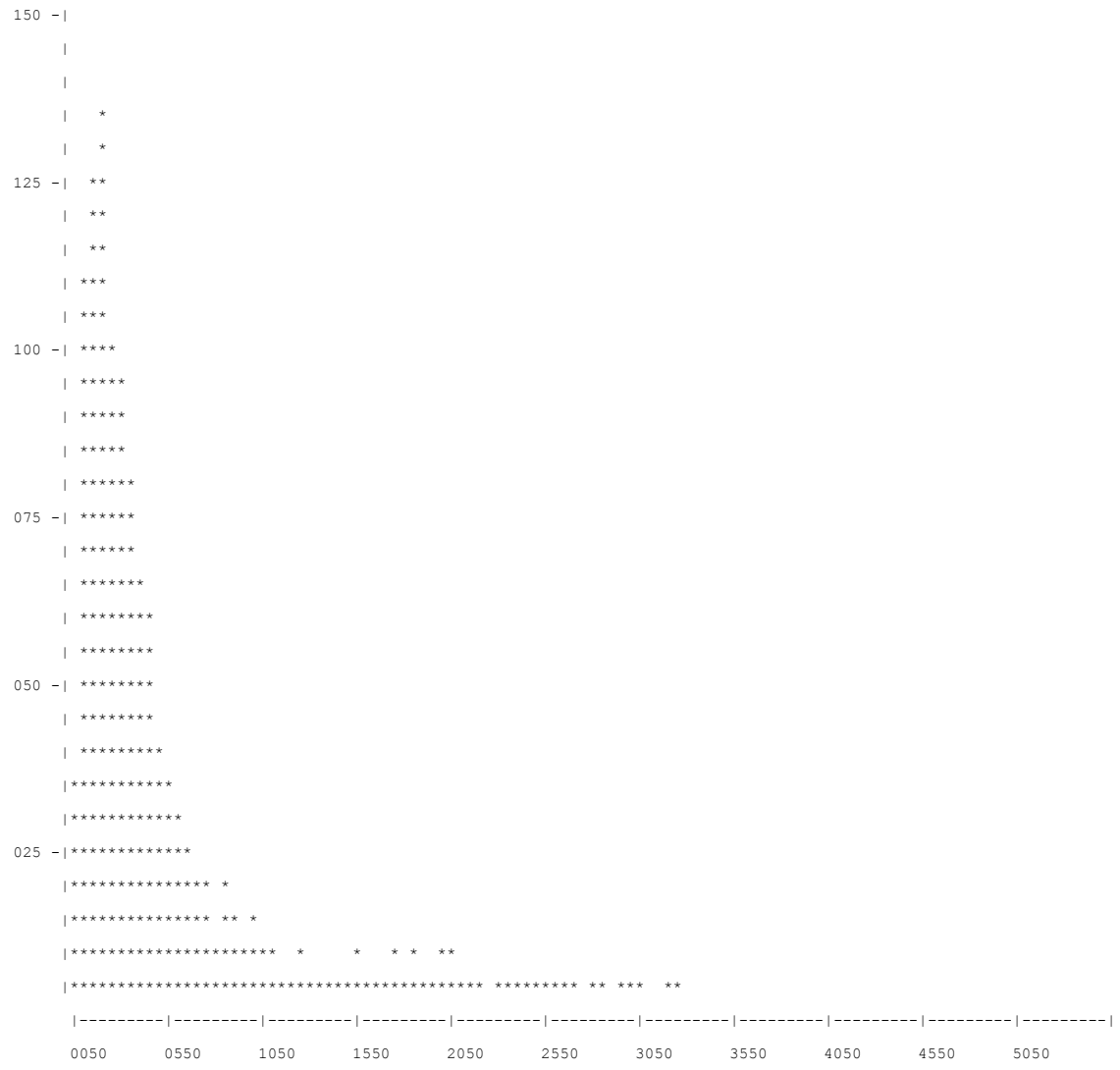
```
 2450 - 2499: 3.
 2400 - 2449: 5.
 2350 - 2399: 3.
 2300 - 2349: 4.
 2250 - 2299: 0.
 2200 - 2249: 1.
 2150 - 2199: 2.
 2100 - 2149: 4.
 2050 - 2099: 6.
 2000 - 2049: 8.
 1950 - 1999: 4.
 1900 - 1949: 2.
 1850 - 1899: 7.
 1800 - 1849: 4.
 1750 - 1799: 7.
 1700 - 1749: 1.
 1650 - 1699: 2.
 1600 - 1649: 4.
 1550 - 1599: 7.
 1500 - 1549: 5.
 1450 - 1499: 5.
 1400 - 1449: 5.
 1350 - 1399: 3.
 1300 - 1349: 2.
 1250 - 1299: 7.
 1200 - 1249: 5.
 1150 - 1199: 4.
 1100 - 1149: 7.
 1050 - 1099: 8.
 1000 - 1049: 13.
 0950 - 0999: 9.
 0900 - 0949: 15.
 0850 - 0899: 16.
 0800 - 0849: 7.
 0750 - 0799: 16.
 0700 - 0749: 19.
 0650 - 0699: 23.
 0600 - 0649: 27.
 0550 - 0599: 35.
 0500 - 0549: 37.
 0450 - 0499: 60.
 0400 - 0449: 63.
 0350 - 0399: 78.
 0300 - 0349: 95.
 0250 - 0299: 100.
 0200 - 0249: 134.
 0150 - 0199: 125.
 0100 - 0149: 110.
 0050 - 0099: 32.

 Count: 1165, Mean: 556.5322, Standard Deviation: 2898.9293, Modal value: 224.0000 (f=134),
Median: 324.0000.
 Low outliers: 23, 24, 25, 23, 24, 23.
 High outliers: 79674, 95846, 243545.
```

The first column `3250 - 3299:` shows the bounds of the interval of times that were counted in the bucket being displayed, in this case from 3,250 milliseconds up to 3,299 milliseconds inclusive. The second column shows the frequency with which timings fell within the bucket being listed, in this case 1.

The statistics line shows the included count of data points, the mean, the standard deviation, the modal and median values of the data set.

The two lines at the bottom of the table show any times that were excluded from the table because they were considered too low or too high.

The –C switch (Chart) will output the statistics as shown in the tables in the form of a frequency histogram.



The Y axis shows the frequency and the X axis shows the timing value.

3. Installing DXIPA

3.1 Building the DXIPA Command Processor

The DXIPA 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 DXIPA 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.

Linux:

Build Environment:

Gcc Version: 4.1.2 for i386-redhat-linux.

Running on Redhat Linux 2.6.18-238.12.1.el5PAE #1 SMP Sat May 7 20:37:06 EDT 2011 i686 i686 i386 GNU/Linux

Notes API Version 8.5

Execution Environment:

Redhat Linux 2.6.18-238.12.1.el5PAE #1 SMP Sat May 7 20:37:06 EDT 2011 i686 i686 i386 GNU/Linux

Domino Server 8.5.1 FP3.

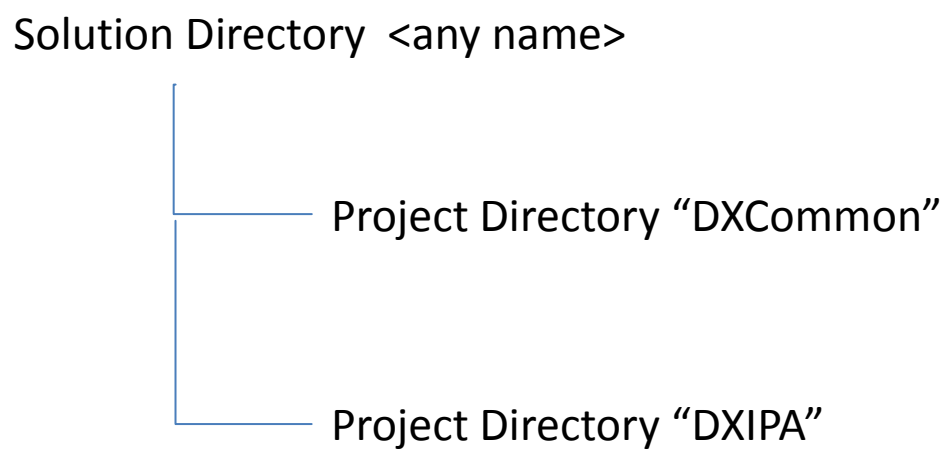
Note: Execution environments from Domino 7.0.x through 8.5.x are regularly used.

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”

Linux:

The DXCommon kernel is supplied as a gzipped archive (.tar.gz). 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.

File ownership and access settings should be adjusted according to your local policies.

As an example.

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

ln -s /usr/include/DXCommon-3.12.0 DXCommon

3.1.5 Installing the DXIPA Sources

Windows:

The DXIPA sources are supplied as a zipped archive (.zip). Create an empty project called “DXIPA” 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

DXIPA.h

Source Files

AppRunSettings.cpp

DXIPA.cpp

Linux:

The DXIPA sources are supplied as a gzipped archive (.tar.gz). Create the “DXIPA” project directory within the <solution directory> unpack the contents of the archive into that directory.

File ownership and access settings should be adjusted according to your local policies.

3.1.6 Build Settings

Windows:

Add each source and header file that is used from the DXCommon kernel to the DXIPA project. To populate the individual filter 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\Platform\PlatBase.h

DXCommon\Platform\WCompat.h

DXCommon\RunSettings.h

DXCommon\Misc\AutoScaleFrequencyAnalyser.h

DXCommon\Threads\ThreadStructs.h

Source Files

DXCommon\RunSettings.cpp

DXCommon\Misc\AutoScaleFrequencyAnalyser.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		Dbghelp.lib Psapi.h

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.

Linux:

A makefile is supplied in the source distribution of DXIPA. The makefile is listed below along with any specific notes. The Makefile supports the following invocation models.

make DXIPA

This form of the command will build any object modules that are out of date and re-link the executable.

make rebuild DXIPA

This form of the command will force a rebuild of all object modules and re-link the executable.

make rebuild DXIPA BV=DBG

This form of the command will force a rebuild of all object modules with the _DEBUG define set and will re-link the executable.

```
#
#  Build for DXIPA 1.0.1 -- Build: 03
#
#  Optional targets:
#
#  rebuild     -  force a complete rebuild of the target
#
#  macros:
#
#  BV=DBG      -  Builds the DEBUG variant of the target
#

TARGET = DXIPA

#  Define the primary source files

SOURCES = $(TARGET).cpp
SOURCES += AppRunSettings.cpp
SOURCES += ../DXCommon/Misc/AutoScaleFrequencyAnalyser.cpp
HEADERS = $(TARGET).h
HEADERS += AppRunSettings.h
HEADERS += ../DXCommon/Platform/PlatBase.h
HEADERS += ../DXCommon/Platform/WCompat.h
HEADERS += ../DXCommon/Misc/AutoScaleFrequencyAnalyser.h

#  Define the Object Lists

OBJECTS = $(TARGET).o
OBJECTS += AppRunSettings.o
OBJECTS += AutoScaleFrequencyAnalyser.o

#  Define the build options

CC = g++
CCOPTS = -c -march=i486
LINKOPTS = -o DXIPA
INCDIR = $(LOTUS)/notesapi/include
DEFINES = -DUNIX -DLINUX -DHANDLE_IS_32BITS
LIBS = -lc

#  Rules to build DEBUG or release targets

$(TARGET): $(OBJECTS)
        $(CC) $(LINKOPTS) $(OBJECTS) -Wl $(LIBS)
$(OBJECTS): $(SOURCES) $(HEADERS)
ifeq ($(BV), DBG)
        $(CC) $(CCOPTS) $(DEFINES) -D_DEBUG -I$(INCDIR) $(SOURCES)
else
        $(CC) $(CCOPTS) $(DEFINES) -I$(INCDIR) $(SOURCES)
endif

#  Phony target for forcing a complete rebuild

.PHONY : rebuild
rebuild:
```

```
-rm *.o
-rm DXIPA
```

Notes:

3.1.7 Building and Deploying the Application

Windows:

Select “Build” and then “Build DXIPA”.

The DXIPA application does not activate a Notes runtime so it can be run from anywhere.

Linux:

Specify one of the following “make” commands to build the application.

make DXIPA

This form of the command will build any object modules that are out of date and re-link the executable.

make rebuild DXIPA

This form of the command will force a rebuild of all object modules and re-link the executable.

make rebuild DXIPA BV=DBG

This form of the command will force a rebuild of all object modules with the _DEBUG define set and will re-link the executable.

The DXIPA application does not activate a Notes runtime so it can be run from anywhere.

4. Running DXIPA

The DXIPA application can be run from anywhere.

DXIPA <ipr file name> [-D][-S][-C]

Where:

<ipr file name> is the name of an instrumentation package recording file (.ipr).

<options> are the command line options for the applications. (see below).

4.1.1 Command Line Options

The following command line options are available with RDBCreate

Option	Meaning
-D	Produces a dump listing of each record in the recording file.
-S	Produces a frequency analysis table and statistical output.
-C	Produces Chart (Histogram) output.