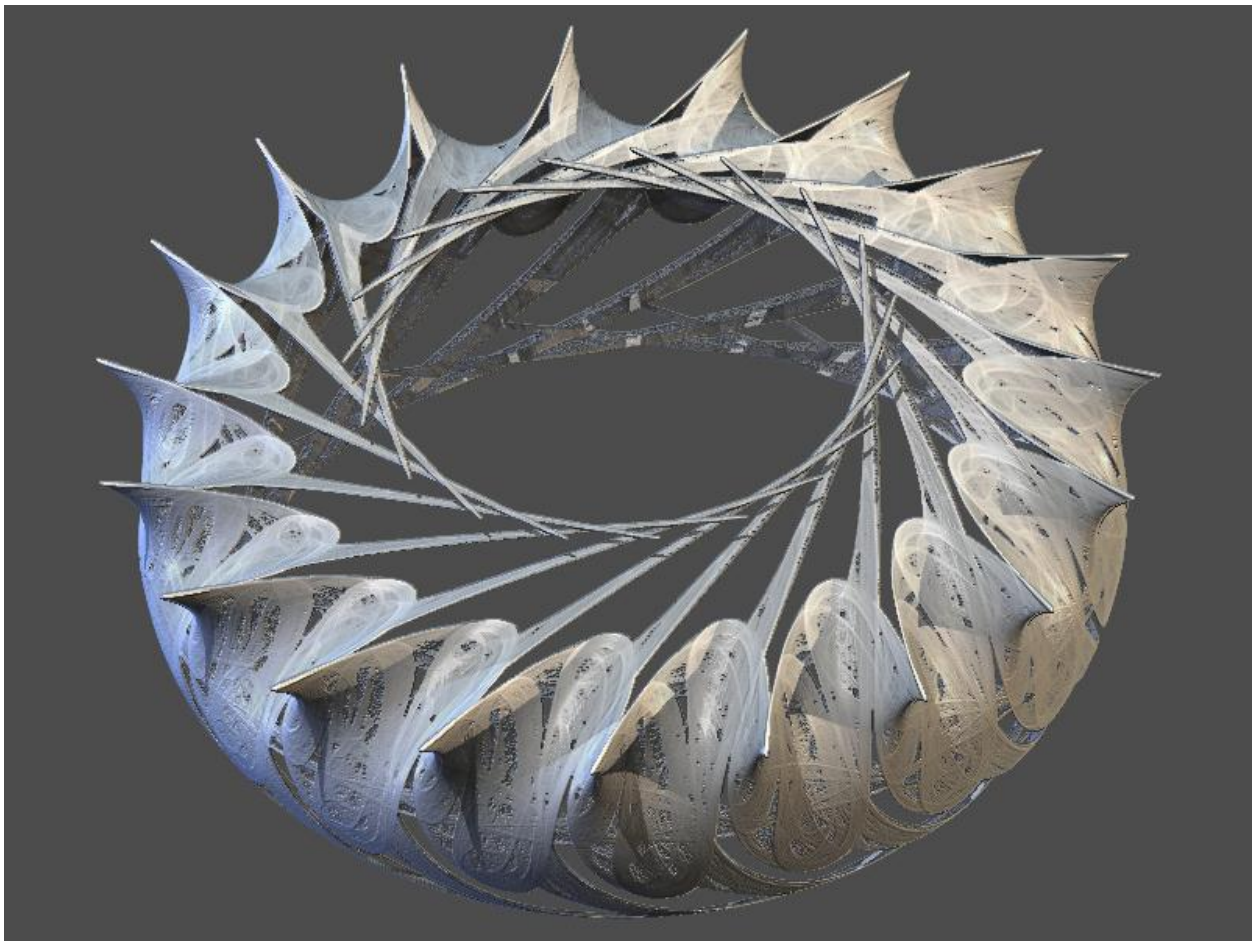
DX Debugging Tools

Using SPZap 1.3



Author: Ian Tree
Owner: HMNL b.*v.*
Customer: Public
Status: Final
Date: 16/04/2012 13:58
Version: 1.3.0

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: 16/04/2012 13:58	Date of next revision *None*

Revision Number	Revision Date	Summary of Changes	Changes marked
1.0.0	14/02/10	Initial Base Version	No
1.3.0	16/04/12	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 SPZap

SPZap (pronounced “super zap”) is a command line file patching utility. The name and some of the commands are an homage to the IBM AMASPZAP (later alias IMASPZAP) utility that was introduced with the MVS operating system. The utility reads commands from the standard input stream (STDIN) and writes the result of the commands to the standard output stream (STDOUT). The utility is designed to operate on any file and assumes that the file is structured as a number of variable sized pages.

The utility has commands to load pages of the file inspect, verify, update and search those pages.

The utility continues to read commands from the input stream until EOF (Ctrl-T on Windows Ctrl-D on Linux) is encountered or a soft End-Of-File, character sequence “/*” is read. If input is redirected from a file then EOF on that file terminates the application.

2. Using SPZap

The SPZap file patching utility is invoked from the command line specifying the name of the file that is to be “zapped” and a default page size.

Syntax:

SPZap <file name> -P:<page size>

<file name> specified the file that is to be “zapped”.

The –P:<page size> may specify the size in decimal, hexadecimal and optionally kilobytes. The following specifications all specify a page size of 4096 bytes, -P:4096, -P:0x1000 and -P:4k (and even pointlessly –P:0x04k). If no default page size is specified then 4k (4096) is used.

Once started, and assuming that the specified file name exists, then commands can be entered from the console or of course redirected from any file or pipe.

All output is written to the standard output stream which can also be redirected to any file or pipe. Every command (or line) that is read from the input is echoed to the standard output stream. Any lines that are entered that do not start with a recognised command are echoed and ignored so the command stream may be decorated with any desired comments.

The utility can have more than one page of file data loaded at any one time, the loaded pages are maintained in a stack. Commands can only act on the page of file data that is at the “top’ of the stack, this is referred to as the “current page”.

2.1 SPZap Commands

This section describes each of the commands that can be entered in the SPZap command stream and the expected outputs.

2.1.1 The PAGESIZE Command

The PAGESIZE command is used to change the size of the current page.

Syntax:

PAGESIZE <page size>

The page size can be specified in decimal or hexadecimal and optionally kilobytes, the following command variations will all change the size of the current page to 4096 bytes.

PAGESIZE 4096

PAGESIZE 0x1000

PAGESIZE 4k

2.1.2 The PAGE Command

The PAGE command is used to load a page of data from the file into memory from the specified offset within the file.

Syntax:

PAGE <file offset> | #<page number>|@<indirection offset>

The file offset may be specified in decimal or hexadecimal long or short format. The size of the page loaded is determined from the current page size. The following command variations will all load a page of data from offset 1024 within the file.

PAGE 1024

PAGE 0x0400

PAGE 0:1024

PAGE 0x00000000:00000400

A message is written to the output showing how much data was loaded from where in the file.

```
ZAP0110I: Page command completed 4096 bytes loaded from offset 0x00000000:00000400.
```

A short page (smaller than the current page size) may be loaded from the end of the file. Any attempt to specify an offset that is past the end of the file will result in an error.

As an alternative to specifying a file offset a page number can be supplied a value of 1 or zero specifies the first page in the file (i.e. offset 0). The offset is computed as (page number – 1) * page size.

Another option is to use indirection, this is dealt with later under “Advanced SPZap Commands”.

2.1.3 The VER Command

The VER command is used to verify the contents of a specified offset within the current page. If a VER command fails because the contents were not as expected then the page is marked so that it cannot be written back to the file. This is common practice with patch utilities to firstly check that the contents of a file are as expected prior to applying a change. The command will also fail if the offset and length of the value would cause the compare to be done past the end of the current page.

Syntax:

VER <offset> <hexadecimal string>

The data in the current page at the specified offset is compared with the value supplied in the hexadecimal string, if the values do not match then the command fails and the page cannot be written back to the file.

The offset can be supplied in decimal or hexadecimal. The following variants of the command will check for a value of F030 at offset 32 in the currently loaded page.

VER 32 F030

VER 0x20 F030

If the command succeeds then a message is written to the output.

```
ZAP0112I: Verify operation succeeded (2 bytes compared).
```

If the command fails then this is also indicated in a message written to the output along with a dump of the data area that was being verified.

```
ZAP0113E: Verify operation failed.
+---- Start of Dump: Verify Failed -------------- 2 (0x0002) bytes @002F2EF2 --------------
0000: EC 15 00 00 00 00 00 00   00 00 00 00 00 00 00 00  :........ ........
+---- End of Dump: Verify Failed -------------- 2 (0x0002) bytes @002F2EF2 --------------
```

2.1.4 The REP Command

The REP command is used to change the content of a specified offset within the current page, the change is only applied to the in-memory copy of a page the WRITE command must be used to commit a changed page to the file.

Syntax:

REP <offset> <hexadecimal string>

The data at the specified offset within the current string is replaced with the value supplied in the hexadecimal string.

The offset can be supplied in decimal or hexadecimal. The following variants of the command will set a value of F030 at offset 32 in the currently loaded page.

REP 32 F030

REP 0x20 F030

If the data can be replaced then a message is written to the output.

```
ZAP0116I: Replace operation succeeded (2 bytes updated).
```

If a previous VER command has failed on the current page then the replace operation is not allowed, this is also indicated in a message.

```
ZAP0107W: This patch cannot be applied because a previous verify has failed.
```

2.1.5 The SEARCH Command

The SEARCH command is used to locate an occurrence of the value supplied in the hexadecimal string in the current page. The search starts at the supplied offset.

Syntax:

SEARCH <offset> <hexadecimal string>

The offset of the first occurrence of the value in the hexadecimal string that can be found in the current page starting at the given offset will be reported.

The offset can be supplied in decimal or hexadecimal. The following variants of the command will locate a value of F030 starting at offset 32 in the currently loaded page.

SEARCH 32 F030

SEARCH 0x20 F030

If the search string can be located then the offset is reported in a message written to the output stream.

```
ZAP0124I: The search string was located at offset 268 (0x010C) in the current page.
```

If the search string could not be located this is indicated by a message written to the output stream.

```
ZAP0125I: The search string was not found in the current page.
```

2.1.6 The DUMP Command

The DUMP command is used to dump the contents of all or part of the current page to the output stream, the dump shows the content in both hexadecimal and character format.

Syntax:

DUMP [<offset>[<size>]]

The contents of the current page are dumped to the output stream, if an offset is specified then the content of the page from the specified offset to the end of the page is dumped. If an offset and a size is specified then <size> bytes of content from the specified offset is dumped.

Both the offset and the size may be specified in decimal or hexadecimal, the following variants of the command will all dump 256 bytes of the current page starting at offset 80.

DUMP 80 256

DUMP 80 0x0100

DUMP 0x0050 0x0100

The DUMP output is shown below.

```
DUMP
DUMP
+---- Start of Dump: Page# 3 - Offset: 0x00000000:00000000 -------------- 512 (0
x0200) bytes @00272EC0 --------------
0000: 1A 00 00 04 00 00 2B 00   00 00 15 08 27 00 37 79  :......+. ....'.7y
0010: 25 C1 00 00 20 02 00 00   E2 02 00 00 85 01 00 00  :%....... ........
0020: 00 00 8A 0C 25 00 DE 79   25 C1 01 FF 40 42 00 00  :....%..y %...@B..
0030: 00 00 00 00 00 00 00 00   00 00 00 00 01 00 A0 16  :........ ........
0040: 00 00 F6 08 00 00 62 23   00 00 55 01 00 10 00 20  :......b# ..U.....
0050: 00 10 00 01 00 00 01 00   00 40 00 00 00 00 00 00  :........ .@......
0060: 00 00 00 00 01 00 00 00   12 07 00 00 CA 00 00 00  :........ ........
0070: 7C 01 00 00 00 00 00 00   00 00 00 00 00 00 00 00  :|....... ........
0080: 00 00 00 00 00 00 00 00   8A 0C 25 00 DE 79 25 C1  :........ ..%..y%.
0090: FC 00 37 00 F8 79 25 C1   00 00 00 00 00 00 00 00  :..7..y%. ........
00A0: 00 00 00 00 00 00 00 00   00 00 00 00 CB 08 27 00  :........ ......'.
00B0: 37 79 25 C1 15 08 27 00   37 79 25 C1 00 00 5A 00  :7y%...'. 7y%...Z.
00C0: FB 00 37 00 80 79 25 C1   44 42 49 20 47 72 6F 75  :..7..y%. DBI.Grou
00D0: 70 20 54 65 73 74 0A 23   32 44 42 49 32 30 30 00  :p.Test.# 2DBI200.
00E0: EF 47 00 60 44 37 08 06   00 00 00 00 00 00 00 00  :.G.`D7.. ........
00F0: 00 00 00 00 B3 18 97 02   2F 00 00 00 03 00 00 00  :........ /.......
0100: 42 3D 00 60 01 00 00 00   9F 12 00 60 B8 18 97 02  :B=.`.... ...`....
0110: 00 00 00 00 B3 18 97 02   2C E6 9D 02 21 17 00 00  :........ ,...!...
0120: 00 00 00 00 5C 1A 18 00   DB 40 00 60 4C 1A 18 00  :....\... .@.`L...
0130: FE 11 00 60 B8 18 97 02   01 00 00 00 60 1A 18 00  :...`.... ....`...
0140: A4 14 00 60 B8 18 97 02   00 00 00 00 00 00 00 00  :...`.... ........
0150: 00 00 00 00 2A 01 00 00   1A 01 00 00 16 01 00 00  :....*... ........
0160: 12 01 00 00 00 00 00 00   00 00 00 00 00 00 00 00  :........ ........
0170: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  :........ ........
0180: 00 00 00 00 00 00 00 00   8E 02 00 00 00 00 00 00  :........ ........
0190: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  :........ ........
01A0: 00 00 00 00 00 00 00 00   00 00 00 00 06 01 00 00  :........ ........
01B0: 00 00 00 00 DE 02 00 00   0A 01 00 00 00 00 00 00  :........ ........
01C0: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  :........ ........
01D0: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  :........ ........
01E0: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  :........ ........
01F0: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  :........ ........
+---- End of Dump: Page# 3 - Offset: 0x00000000:00000000 -------------- 512 (0x0
200) bytes @00272EC0 --------------
```

The page number shown in the header and trailer lines of the dump indicates the sequence number of PAGE commands that have been executed. The offset is the offset of the start of the page. Sixteen bytes are dumped in each output line, two groups of 8 bytes each are dumped in hexadecimal and then in character format. In the character format dump blank and unprintable characters are replaced by a dot "." Character.

```
DUMP 0x00d0 0x0030
DUMP 0x00d0 0x0030
+---- Start of Dump: Page# 3 - Offset: 0x00000000:00000000 (+208) --------------
 48 (0x0030) bytes @00272F90 --------------
0000: 70 20 54 65 73 74 0A 23   32 44 42 49 32 30 30 00  :p.Test.# 2DBI200.
0010: EF 47 00 60 44 37 08 06   00 00 00 00 00 00 00 00  :.G.`D7.. ........
0020: 00 00 00 00 B3 18 97 02   2F 00 00 00 03 00 00 00  :........ /.......
+---- End of Dump: Page# 3 - Offset: 0x00000000:00000000 (+208) -------------- 4
8 (0x0030) bytes @00272F90 --------------
```

In the case of a DUMP of only a part of the page the offset still shows the offset of the start of the page followed by the offset within the page in brackets.

2.1.7 The BITDUMP Command

The BITDUMP command is used to dump the contents of all or part of the current page to the output stream, the dump shows the content in both hexadecimal and bit string format.

Syntax:

BITDUMP [<offset>[<size>]]

The contents of the current page are dumped to the output stream, if an offset is specified then the content of the page from the specified offset to the end of the page is dumped. If an offset and a size is specified then <size> bytes of content from the specified offset is dumped.

Both the offset and the size may be specified in decimal or hexadecimal, the following variants of the command will all dump 256 bytes of the current page starting at offset 80.

BITDUMP 80 256

BITDUMP 80 0x0100

BITDUMP 0x0050 0x0100

The BITDUMP output is shown below.

```
BITDUMP 0x00d0 0x0030
BITDUMP 0x00d0 0x0030
+---- Start of Dump: Page# 3 - Offset: 0x00000000:00000000 (+208) --------------
 48 (0x0030) bytes @00272F90 --------------
0000: 70 20 54 65 73 74 0A 23   :0111 0000 0010 0000 0101 0100 0110 0101 0111 00
11 0111 0100 0000 1010 0010 0011
0008: 32 44 42 49 32 30 30 00   :0011 0010 0100 0100 0100 0010 0100 1001 0011 00
10 0011 0000 0011 0000 0000 0000
0010: EF 47 00 60 44 37 08 06   :1110 1111 0100 0111 0000 0000 0110 0000 0100 01
00 0011 0111 0000 1000 0000 0110
0018: 00 00 00 00 00 00 00 00   :0000 0000 0000 0000 0000 0000 0000 0000 0000 00
00 0000 0000 0000 0000 0000 0000
0020: 00 00 00 00 B3 18 97 02   :0000 0000 0000 0000 0000 0000 0000 0000 1011 00
11 0001 1000 1001 0111 0000 0010
0028: 2F 00 00 00 03 00 00 00   :0010 1111 0000 0000 0000 0000 0000 0000 0000 00
11 0000 0000 0000 0000 0000 0000
+---- End of Dump: Page# 3 - Offset: 0x00000000:00000000 (+208) -------------- 4
8 (0x0030) bytes @00272F90 --------------
```

Each line of the output shows 8 bytes of the content firstly in hexadecimal and then in bit format.

2.2 Advanced SPZap Commands

2.2.1 The PUSH Command

The PUSH command saves the contents of the current page on the page stack and makes a new empty page available as the current page.

Syntax:

PUSH

2.2.2 The POP Command

The POP command discards the current page and restores the last page that was saved on the page stack with a previous PUSH command.

Syntax:

POP

2.2.3 The PAGE @ Command

This variant of the PAGE command loads a page from the file but instead of specifying the file offset of the page to be loaded directly it takes the offset from the value stored at a given offset in the current or last saved page.

Syntax:

PAGE @<offset> <format>

The offset can be supplied in decimal or hexadecimal.

The format specifies the format of the data at the specified offset and must be one of the following values.

WI16, WI24, WI32, WI48, WI56 or WI64 these all specify Intel format words or part word formats.

WC16, WC24, WC32, WC48, WC56 or WC64 these all specify canonical format or par word formats.

WD32 this specifies Relative Granule Address format which is often used in Domino .nsf databases.

The content of the current page at the specified offset is formatted according to the type format specified and formed into a file offset, the page at the calculated offset is then loaded into the current page. If there is no current page loaded then the command uses the last page that was saved on the page stack by a previous PUSH command.

The following sequence of commands would be used to dump two pages, the file offsets of these pages are stored in a third page.

PAGE 0x00000000:00008000

PUSH

PAGE @0x0020

DUMP

POP

PUSH

PAGE @0x0028

DUMP

/*

3. Installing SPZap

3.1 Building the SPZap Command Processor

The SPZap command processor is built on the Domino eXplorer Tools DXCommon kernel, you need to download and install this before you can build the SPZap 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.

Linux:

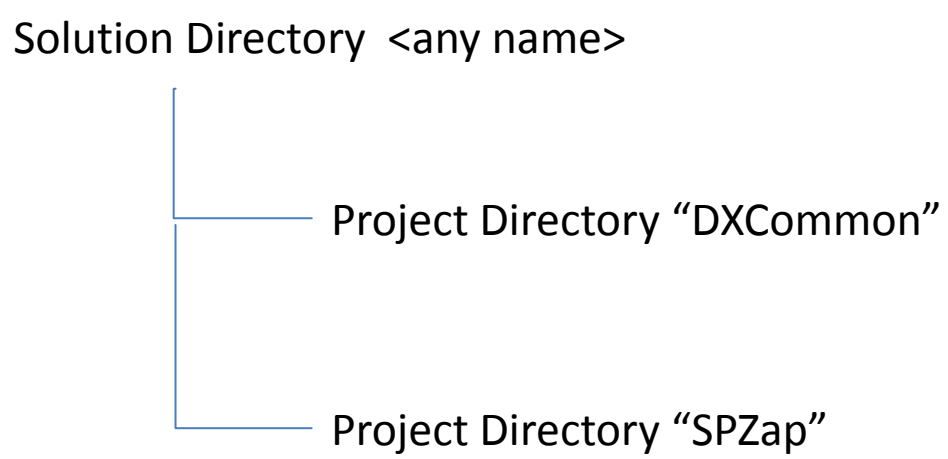
Build Environment:

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

3.1.2 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.

3.1.3 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

3.1.4 Installing the SPZap Sources

Windows:

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

SPZap.h

Source Files

AppRunSettings.cpp

SPZap.cpp

Linux:

The SPZap sources are supplied as a gzipped archive (.tar.gz). Create the “SPZap” 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.

3.1.5 Build Settings

Windows:

Add each source and header file that is used from the DXCommon kernel to the SPZap 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

Source Files

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 SPZap. The makefile is listed below along with any specific notes. The Makefile supports the following invocation models.

make SPZap

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

make rebuild SPZap

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

make rebuild SPZap 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 SPZap 1.3.1 -- Build: 11
#
#  Optional targets:
#
#  rebuild    -  force a complete rebuild of the target
#
#  macros:
#
#  BV=DBG     -  Builds the DEBUG variant of the target
#

TARGET = SPZap

#  Define the primary source files

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

#  Define the Object Lists

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

#  Build the complete build lists

CC = g++
CCOPTS = -c -march=i486
LINKOPTS = -o $(TARGET)
DEFINES = -DUNIX -DLINUX -DHANDLE_IS_32BITS
LIBS = -lg++

#  Rules to build the release or debug targets

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

#  Phony target for forcing a complete rebuild
```

```
.PHONY : rebuild
rebuild:
	-rm *.o
	-rm $(TARGET)
```

Notes:

3.1.6 Building and Deploying the Application

Windows:

Select “Build” and then “Build SPZap”.

The SPZap 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 SPZap

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

make rebuild SPZap

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

make rebuild SPZap 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 SPZap application does not activate a Notes runtime so it can be run from anywhere.

4. Running SPZap

The SPZap application can be run from anywhere.

SPZap <file name> -P:<page size>

Where:

<file name> specified the file that is to be “zapped”.

<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 SPZap

Option	Meaning
-P:<size>	Specifies the size in decimal, hexadecimal and optionally kilobytes. The following specifications all specify a page size of 4096 bytes, -P:4096, -P:0x1000 and -P:4k (and even pointlessly –P:0x04k). If no default page size is specified then 4k (4096) is used.