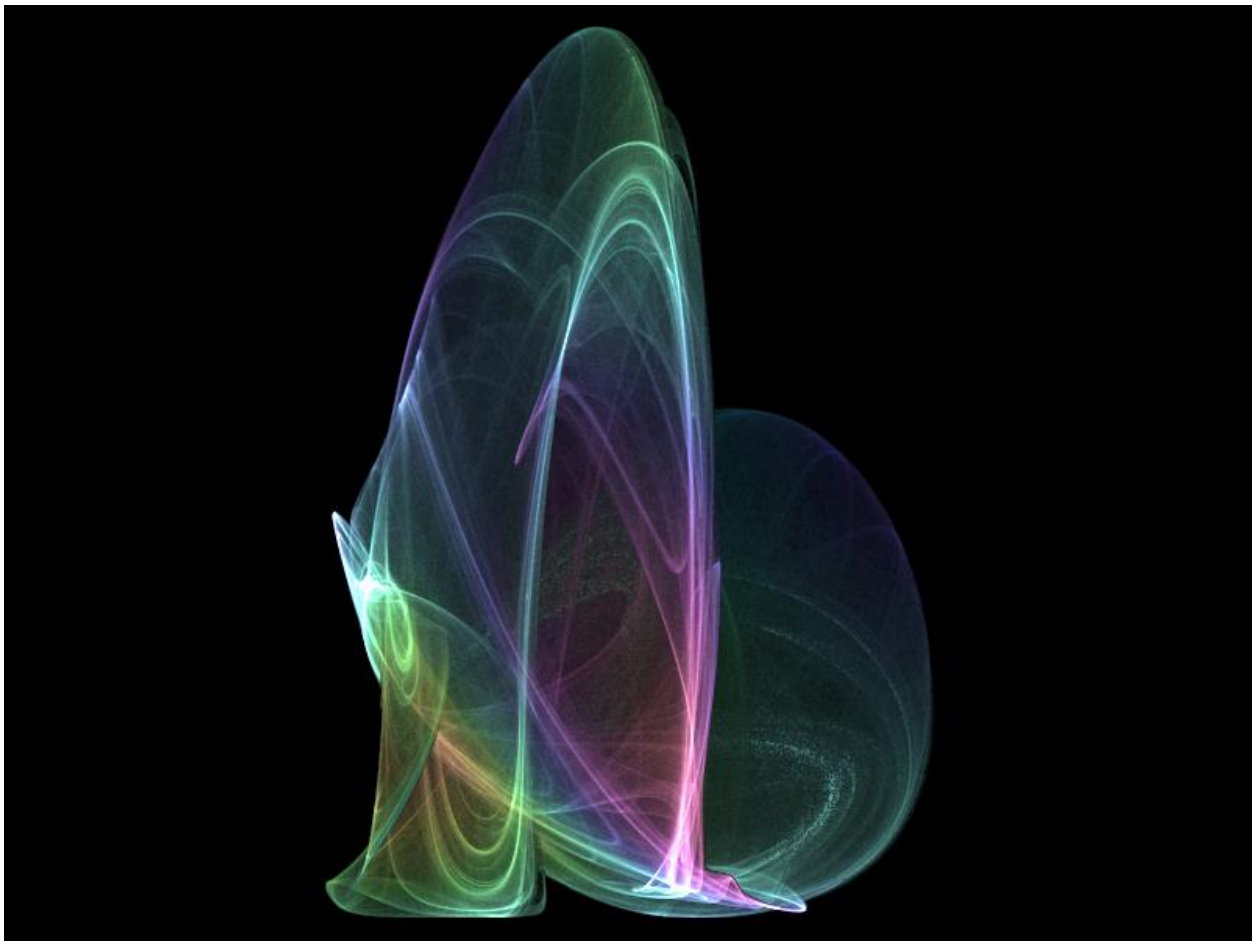
DX Tools

Using RDBCreate 2.1



Author: Ian Tree
Owner: HMNL b.*v.*
Customer: Public
Status: Final
Date: 23/03/2012 10:44
Version: 2.1.0

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: 23/03/2012 10:47	Date of next revision *None*

Revision Number	Revision Date	Summary of Changes	Changes marked
2.0.0	14/04/11	Initial Base Version	No
2.1.0	23/03/12	QE Version	No

Acknowlegements

Frontpiece Design was produced by the chaoscope application.

Contents

1. Introduction to RDBCreate

RDBCreate is a command processor that is used to create DXTools databases from a source “Virtual Template” that is available on the internet or a local source. The utility can also be used to update the design of an existing database.

Virtual Templates provide a quick and easy mechanism for distributing the designs of Notes databases without the need for replicating or copying a physical template before creating new databases from the template. Because a Virtual Template can be used from almost anywhere it is possible to have a single point of source for a design, this ensures that new databases always use the latest available design.

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

RDBCreate is a C++ application constructed on the DXCommon application kernel supporting Domino version from 6.0 through 8.5 on both Widows & Linux server platforms.

2. Virtual Templates

Virtual Templates provide a quick and easy mechanism for distributing the designs of Notes databases without the need for replicating or copying a physical template before creating new databases from the template. Because a Virtual Template can be used from almost anywhere it is possible to have a single point of source for a design, this ensures that new databases always use the latest available design.

2.1 Structure

A Virtual Template consists of a Manifest, this is an XML document that describes the complete payload needed for a Virtual Template and a collection of DXL documents that contain the XML source for each individual design element.

The Manifest then consists of a number of Design Sets, each Design Set defines a number of individual Design Elements. The grouping of Design Elements into Sets allows for easy re-use of groups of Design Elements that are needed to provide particular functionality. Each Design Element describes a Design Item e.g. Form, view, agent and so on and provides the URL to retrieve the DXL for that design element.

The Manifest and the DXL source for each design element are published via HTTP but can be published as files in a file system that is accessible to the RDBCreate application.

2.2 Manifest XML Document

The following example shows a notional XML Manifest document.

```
<?xml version="1.0" encoding="UTF-8"?>
<designmanifest name="Example" version="1.0">
<!--- Images -->
   <designset name="Images">
      <designelement name="hmnl-logo-transparent.gif" type="imageresource">
http://hmnl.nl/HMNL/DX/VTTDepot.nsf/Payloads/qmodify2.imageresource.hmnl-logo-
transparentgif/$File/hmnl-logo-transparentgif.xml
      </designelement>
      <designelement name="$PlusMinus.gif" type="imageresource">
http://hmnl.nl/HMNL/DX/VTTDepot.nsf/Payloads/qmodify2.imageresource.$PlusMinusgif/$File/$PlusMinu
sgif.xml
      </designelement>
   </designset>
<!--- Subforms -->
   <designset name="Subforms">
      <designelement name="FormHeader" type="subform" sign="1">
http://hmnl.nl/HMNL/DX/VTTDepot.nsf/Payloads/qmodify2.subform.FormHeader/$File/FormHeader.xml
      </designelement>
      <designelement name="subAbout" type="subform" sign="1">
http://hmnl.nl/HMNL/DX/VTTDepot.nsf/Payloads/qmodify2.subform.subAbout/$File/subAbout.xml
      </designelement>
   </designset>
<!--- Forms -->
   <designset name="Forms">
      <designelement name="QModify Transaction" type="form" sign="1">
http://hmnl.nl/HMNL/DX/VTTDepot.nsf/Payloads/qmodify2.form.QModifyTransaction/$File/QModifyTransa
ction.xml
      </designelement>
   </designset>
<!--- Views -->
   <designset name="Views">
      <designelement name="Completed Transactions" type="view" sign="1">
http://hmnl.nl/HMNL/DX/VTTDepot.nsf/Payloads/qmodify2.view.CompletedTransactions/$File/CompletedT
ransactions.xml
      </designelement>
      <designelement name="Delayed Transactions" type="view" sign="1">
http://hmnl.nl/HMNL/DX/VTTDepot.nsf/Payloads/qmodify2.view.DelayedTransactions/$File/DelayedTrans
actions.xml
      </designelement>
```

```
   </designset>
</designmanifest>
```

2.2.1 designmanifest Node

The designmanifest is the top level node in the XML document and encloses the complete manifest contents. The name and version attributes are arbitrary string values that are used to create a default database title for any new database created from the manifest.

2.2.2 designset Node

The designmanifest node contains any number of designset nodes. A designset node is an arbitrary collection of design elements. The name attribute is a string that is used in reporting messages by the RDBCreate application. Although the example document shows design sets containing elements of the same type this is not a requirement.

2.2.3 designelement Node

The designelement node describes a single design element that will be applied to a database. The node contains the HTTP URL or local file name of the DXL source for the design element, the URL or filename must be specified on a separate line of the XML document from the start or end of the node. The following attributes are available on a designelement node.

2.2.3.1 name

The name attribute is mandatory and **MUST** specify a name or alias for the design element that can be resolved in the database design collection.

2.2.3.2 type

The type attribute is mandatory and specifies the type of design element that is to be imported the type can specify any of the following values.

"PAGE", "VIEW", "FORM", "SUBFORM", "FRAMESET", "AGENT", "IMAGERESOURCE", "SHAREDFIELD", "SHAREDACTION", "LIBRARY", "FOLDER", "OUTLINE", "DBICON", "SHAREDCOLUMN", "HELPABOUT", "HELPUSING".

The values are self-explanatory.

2.2.4 sign

The sign attribute is optional a value of “1” specifies that the design element should be signed after it is imported.

2.3 Design Element DXL Sources

The design element exports are, with the exception of a DBICON export which is described later, DXL documents as produced by one of the native Domino DXL engines.

2.3.1 DBICON DXL Source

The DBICON DXL source is a specially crafted DXL document that contains at the database the application launch options that are to be used in the database (the <launchsettings> node) and an image resource design element called “DX-Dummy-Carrier” that carries an arbitrary image (which is not used) and an item called “IconBitmap” containing the database Icon that is to be used. The easiest way to

create one of these is either with the DCLoader application from the DX Tools suite or to download an existing one and hand edit it to replace the launch settings and the IconBitmap item.

2.4 Customising a Virtual Template

The steps below illustrate how the example manifest could be customised to replace one of the images that are displayed in the header of the form to show a locally badged version with a custom image.

2.4.1 Create Custom Image DXL

Create a DXL export of the selected custom image resource as a file on the local computer (c:\Notes\Exports\CustomImage.dxl). Edit the DXL file to change the name of the image resource to "hmnl-logo-transparent.gif" as this is the name that is used in the native design of the database.

2.4.2 Download the Manifest

Open the manifest URL in a browser and save the XML to disk (c:\Notes\Exports\MyManifest.xml).

2.4.3 Customise the Manifest

Replace the URL of the DXL source for the image resource with the file name of the local custom image.

```
      <designelement name="hmnl-logo-transparent.gif" type="imageresource">
http://hmnl.nl/HMNL/DX/VTTDepot.nsf/Payloads/qmodify2.imageresource.hmnl-logo-
transparentgif/$File/hmnl-logo-transparentgif.xml
      </designelement>
```

Becomes..

```
      <designelement name="hmnl-logo-transparent.gif" type="imageresource">
C:\Notes\Exports\CustomImage.dxl
      </designelement>
```

2.4.4 Create a Database Using the Custom Manifest

Run RDBCreate specifying c:\Notes\Exports\MyManifest.xml as the manifest source.

3. Installing RDBCreate

3.1 Building the RDBCreate Command Processor

The RDBCreate 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 RDBCreate 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.

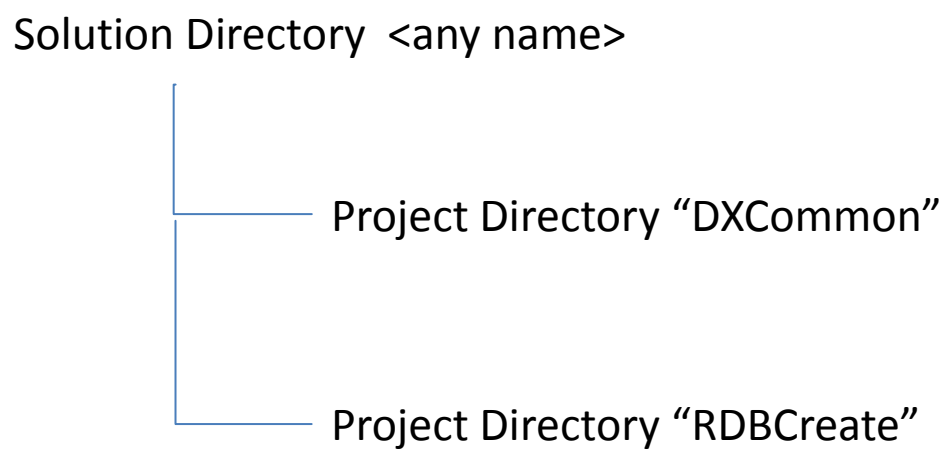
Note: Execution environments from Domino 7.0.x through 8.5.x are regularly used.

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 RDBCreate Sources

Windows:

The RDBCreate sources are supplied as a zipped archive (.zip). Create an empty project called "RDBCreate" 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

RDBCreate.h

Source Files

AppRunSettings.cpp

RDBCreate.cpp

Linux:

The RDBCreate sources are supplied as a gzipped archive (.tar.gz). Create the "RDBCreate" 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 RDBCreate project. It is convenient to add these source and header files into subsets that can then be copied into other projects, in Visual Studio 2005 these collections are referred to as "filters". To create a filter right-click on either the "Header Files" folder or the "Source Files" folder and select "Add" then "New Filter". The following filters are convenient to use in the RDBCreate project "Designer" and "Runtime" these filters should be created in both the "Source Files" and "Header Files" folders. 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\Designer

DXCommon\DM\DesignElement.h

DXCommon\DM\DesignManager.h

DXCommon\DM\DesignManifest.h

DXCommon\DM\DesignManifestLine.h

DXCommon\DM\DesignSet.h

DXCommon\DXResource.h

DXCommon\DXResourceLoader.h

DXCommon\DXUCItem.h

DXCommon\DXUploadContext.h

Header Files\Runtime

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\Designer

DXCommon\DM\DesignElement.cpp

DXCommon\DM\DesignManager.cpp

DXCommon\DM\DesignManifest.cpp

DXCommon\DM\DesignManifestLine.cpp

DXCommon\DM\DesignSet.cpp

DXCommon\DXResource.cpp

DXCommon\DXResourceLoader.cpp

DXCommon\DXUCItem.cpp

DXCommon\DXUploadContext.cpp

Source Files\Runtime

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 winhttp.lib	notes.lib winhttp.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.

Linux:

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

make RDBCreate

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

make rebuild RDBCreate

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

make rebuild RDBCreate 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 RDBCreate 2.1.0 -- Build: 59 -- DXCommon 3.12.0
#
#  Optional targets:
#
#  rebuild     -  force a complete rebuild of the target
#
#  macros:
#
#  BV=DBG      -  Builds the DEBUG variant of the target
#

TARGET = RDBCreate

#  Define the primary source files
SOURCES = $(TARGET).cpp
SOURCES += AppRunSettings.cpp
HEADERS = $(TARGET).h
HEADERS += AppRunSettings.h

#  Define Core modules to build

CORE_SOURCES = ../DXCommon/APIPackages.cpp
CORE_SOURCES += ../DXCommon/DXException.cpp
CORE_SOURCES += ../DXCommon/ExecEnvironment.cpp
CORE_SOURCES += ../DXCommon/ElapsedTimer.cpp
CORE_SOURCES += ../DXCommon/RunSettings.cpp
CORE_HEADERS = ../DXCommon/APIPackages.h
CORE_HEADERS += ../DXCommon/DXException.h
CORE_HEADERS += ../DXCommon/ExecEnvironment.h
CORE_HEADERS += ../DXCommon/ElapsedTimer.h
CORE_HEADERS += ../DXCommon/RunSettings.h
CORE_HEADERS += ../DXCommon/DXGlobals.h
CORE_HEADERS += ../DXCommon/Platform/PlatBase.h
CORE_HEADERS += ../DXCommon/Platform/NotesBase.h

#  Define the Designer modules to build

DESIGN_SOURCES = ../DXCommon/DM/DesignElement.cpp
DESIGN_SOURCES += ../DXCommon/DM/DesignManager.cpp
DESIGN_SOURCES += ../DXCommon/DM/DesignManifest.cpp
DESIGN_SOURCES += ../DXCommon/DM/DesignManifestLine.cpp
DESIGN_SOURCES += ../DXCommon/DM/DesignSet.cpp
DESIGN_SOURCES += ../DXCommon/DXResource.cpp
DESIGN_SOURCES += ../DXCommon/DXResourceLoader.cpp
DESIGN_SOURCES += ../DXCommon/DXUCItem.cpp
DESIGN_SOURCES += ../DXCommon/DXUploadContext.cpp
DESIGN_HEADERS = ../DXCommon/DM/DesignElement.h
DESIGN_HEADERS += ../DXCommon/DM/DesignManager.h
DESIGN_HEADERS += ../DXCommon/DM/DesignManifest.h
DESIGN_HEADERS += ../DXCommon/DM/DesignManifestLine.h
DESIGN_HEADERS += ../DXCommon/DM/DesignSet.h
DESIGN_HEADERS += ../DXCommon/DXResource.h
DESIGN_HEADERS += ../DXCommon/DXResourceLoader.h
DESIGN_HEADERS += ../DXCommon/DXUCItem.h
```

```
DESIGN_HEADERS += ../DXCommon/DXUploadContext.h

#  Define the Object Lists

OBJECTS = $(TARGET).o
OBJECTS += AppRunSettings.o
CORE_OBJECTS = APIPackages.o
CORE_OBJECTS += DXException.o
CORE_OBJECTS += ExecEnvironment.o
CORE_OBJECTS += ElapsedTimer.o
CORE_OBJECTS += RunSettings.o
DESIGN_OBJECTS = DesignElement.o
DESIGN_OBJECTS += DesignManager.o
DESIGN_OBJECTS += DesignManifest.o
DESIGN_OBJECTS += DesignManifestLine.o
DESIGN_OBJECTS += DesignSet.o
DESIGN_OBJECTS += DXResource.o
DESIGN_OBJECTS += DXResourceLoader.o
DESIGN_OBJECTS += DXUCItem.o
DESIGN_OBJECTS += DXUploadContext.o

#  Build the complete build lists

CC = g++
CCOPTS = -c -march=i486
NOTESDIR = $(LOTUS)/notes/latest/linux
LINKOPTS = -o rdbcreate
DEFINES = -DUNIX -DLINUX -DHANDLE_IS_32BITS
INCDIR = $(LOTUS)/notesapi/include
LIBS = -lnotes -lm -lnsl -lpthread -lc -lresolv -ldl -lcurl

#  Rules to build DEBUG or release targets

$(TARGET): $(OBJECTS) $(CORE_OBJECTS) $(DESIGN_OBJECTS)
        $(CC) $(LINKOPTS) $(OBJECTS) $(CORE_OBJECTS) $(DESIGN_OBJECTS) -L$(NOTESDIR) -Wl,-rpath-
link $(NOTESDIR) $(LIBS)
$(OBJECTS): $(SOURCES) $(HEADERS)
ifeq ($(BV), DBG)
        $(CC) $(CCOPTS) $(DEFINES) -D_DEBUG -I$(INCDIR) $(SOURCES)
else
        $(CC) $(CCOPTS) $(DEFINES) -I$(INCDIR) $(SOURCES)
endif
$(CORE_OBJECTS): $(CORE_SOURCES) $(CORE_HEADERS)
ifeq ($(BV), DBG)
        $(CC) $(CCOPTS) $(DEFINES) -D_DEBUG -I$(INCDIR) $(CORE_SOURCES)
else
        $(CC) $(CCOPTS) $(DEFINES) -I$(INCDIR) $(CORE_SOURCES)
endif
$(DESIGN_OBJECTS): $(DESIGN_SOURCES) $(DESIGN_HEADERS)
ifeq ($(BV), DBG)
        $(CC) $(CCOPTS) $(DEFINES) -D_DEBUG -I$(INCDIR) $(DESIGN_SOURCES)
else
        $(CC) $(CCOPTS) $(DEFINES) -I$(INCDIR) $(DESIGN_SOURCES)
endif

#  Phony target for forcing a complete rebuild

.PHONY : rebuild
rebuild:
        -rm *.o
        -rm rdbcreate
```

Notes:

There is currently little difference between the release and debug builds of the RDBCreate application, the debugging helper that adds additional runtime debugging capabilities to the application is only available for the Windows builds. There is an enhancement request to port the debugging helper functionality to Linux.

The executable name is coerced to lower case “rdbcreate”, this is a Domino convention.

3.1.7 Building and Deploying the Application

Windows:

Select “Build” and then “Build RDBCreate”.

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

Linux:

make RDBCreate

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

make rebuild RDBCreate

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

make rebuild RDBCreate 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.

Copy the resulting executable (rdbcreate) to the Notes Executable directory where you want to run the Command Processor. According to your local security policies and Domino install you may need to have an administrator copy the executable and possibly change ownership of the executable.

The default ownership and attributes indicated by the Notes API documentation are as follows.

chown server rdbcreate

chgrp notes rdbcreate

chmod 2555 rdbcreate

4. Running RDBCreate

The RDBCreate application can be loaded on a server or run from a command window on a workstation.

On a server use “load RDBCreate <server name> <database name> <manifest URL> <options>”

Where:

<server name> is the abbreviated name of the server on which you want to create the database, specify “Local” to create the database on the server/workstation where RDBCreate will run.

<database name> is the name of the new database relative to the notes data directory.

<manifest URL> is the HTTP URL or file name of the Virtual Template manifest that is to be applied to the new database.

<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
-V	Sets the logging level to verbose. This provides more detailed logging messages to be written to the application log.
-T[:nn]	Sets the logging level to trace, This is a diagnostic setting that provides detailed logging messages to the application log. The optional “nn” setting restricts tracing to a designated are of the DXCommon kernel code. Refer to the DXGlobals.h header file for values that this setting can take.
-D[:nn]	Sets the logging level to debug, This is a diagnostic setting that provides even more detailed logging messages to the application log. The optional “nn” setting restricts tracing to a designated are of the DXCommon kernel code. Refer to the DXGlobals.h header file for values that this setting can take.
-E	Sets console echo mode on. Application log entries are written to documents in the control database. When console echo mode is on then all application log messages are echoed to the Domino console and the Domino system log database.
-U	Specifies that the design will update the design of an existing database rather than creating a new database.