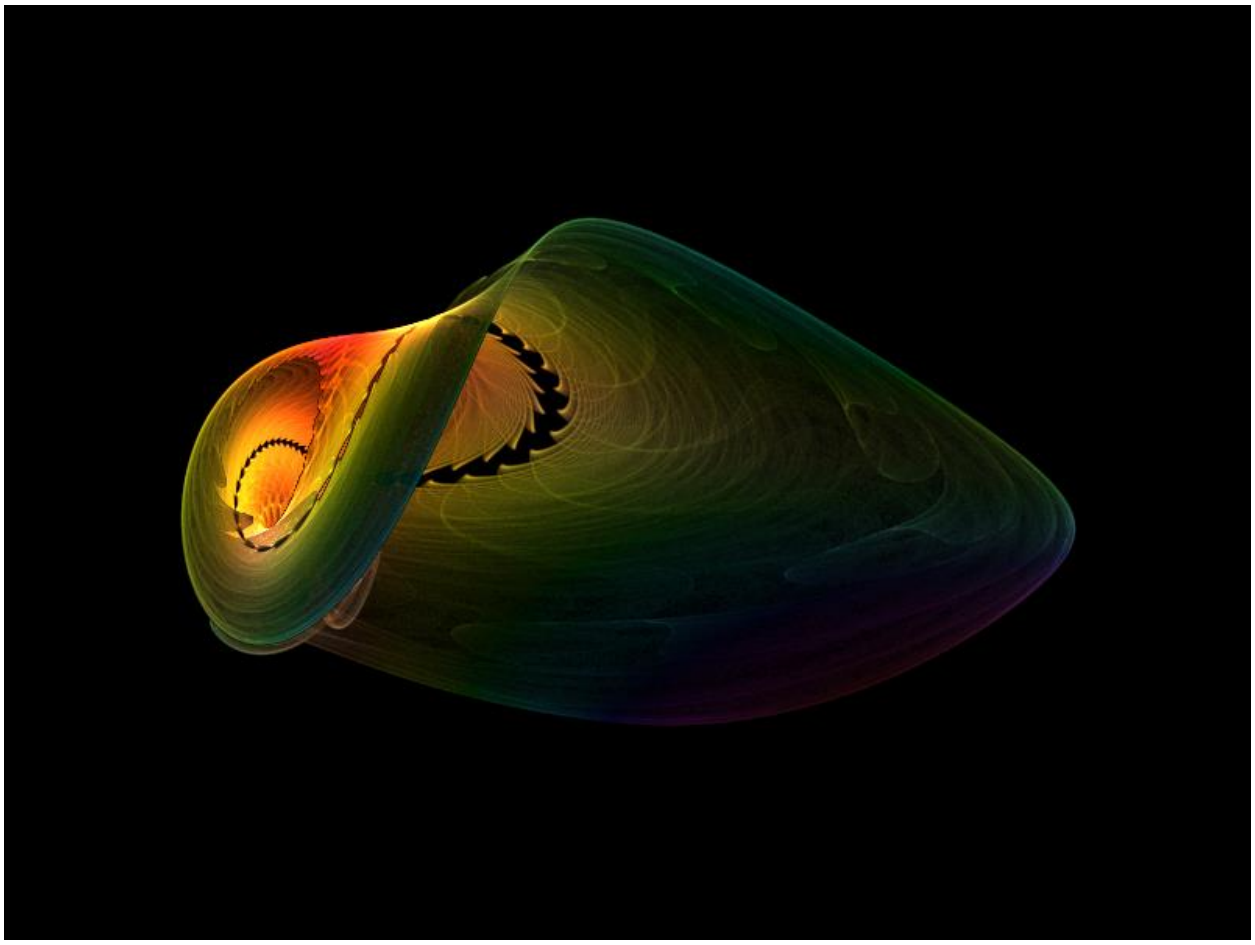
DX Tools

Class Catalogue 3.14



Author: Ian Tree

Owner: HMNL b.v.

Customer: Public

Status: Final

Date: 10/03/2015 16:36

Version: 3.14.0

Disposition: Open Source

Document Usage

Revision History

Date of this revision: 10/03/2015 16:36	Date of next revision *None*

Revision Number	Revision Date	Summary of Changes	Changes marked
0.1	04/02/09	Initial Base Version	No
3.12.0	17/04/12	QE Version	No
3.14.0	10/03/15	Updated for x64 support	No

Acknowlegements

Frontpiece Design was produced by the chaoscope application.

Contents

1. Introduction to DX Classes

The Domino eXplorer (DX) was developed as a means for facilitating the rapid development of tools to be used in projects that involve high volumes of data transformation in Domino databases. DX has been, and continues to be developed for use across a wide range of Domino versions and platforms. The reference platforms are Domino 9.0.x on Windows Server 2003 R2 (32 and 64 bit) and Red Hat Linux 6.6. DX is also used as a research tool to investigate various aspects of Autonomic Systems, in particular Autonomic Throughput Optimisation.

Standardised utilities have also been built around some of the functional DX classes, these are published as “DX Tools” and can save time by providing off-the-shelf processing to be incorporated into complex transformations that need high throughput rates.

DX consists of a set of “Kernel” classes and a collection of “Functional” classes. In this document there is only a short description of the “Kernel” classes as they are dealt with in other documents. This document concentrates on the “Functional” classes and how to use them. Classes that do not expose functionality to applications but are used by the “Functional” classes are documented under the section “Supporting” classes.

2. Kernel Classes

This section provides a brief description of the classes that make up the Domino eXplorer kernel.

The kernel classes are used in a DX application to deliver basic functionality and provide an execution context for the functional classes and custom code.

2.1 ExecEnvironment Class

Header File: DXCommon/ExecEnvironment.h

The ExecEnvironment class is the core of the runtime system for single threaded applications. The class also provides a large collection of convenience methods and linkage to other objects.

The class is instantiated as a singleton object early in the execution lifecycle of an application. The singleton object is widely accessed in an application and passed to the constructors of most DX classes.

Construction of the singleton will initialise the Notes/Domino runtime and establish the appropriate run context for the application providing logging and Message Queues if the application is configured as a Server Add-In task. The application must provide a RunSettings object, or usually a object of a class that extends the RunSettings class to the constructor, this object defines the application configuration.

Throughout this document the singleton is referred to as the “runtime object”.

2.2 MTExecutive Class

Header File: DXCommon/MTX/MTExecutive.h

The MTExecutive class extends the ExecEnvironment class to provide multi-threaded support to applications.

The class is instantiated as a singleton object early in the execution lifecycle of an application. The singleton object is widely accessed in an application and passed to the constructors of most DX classes.

Construction of the singleton will initialise the Notes/Domino runtime and establish the appropriate run context for the application providing logging and Message Queues if the application is configured as a Server Add-In task. The construction of the object will also establish a pool of threads that can be used to execute asynchronous actions as well as the housekeeping threads that provide the multi-threaded execution environment. The application must provide a RunSettings object, or usually a object of a class that extends the RunSettings class to the constructor, this object defines the application configuration. The application can optionally provide a ThreadManagerPolicy object to the comstructor, this policy object allows low-level control over the timings and other constraints in the multi-threading kernel.

Throughout this document the singleton is referred to as the “runtime object”.

2.3 APIPackages Class

Header File: DXCommon/APIPackages.h

This class is constructed by the runtime and is used to translate a Notes API status code into a decorated readable form. Applications will not normally access functions in this class directly instead they will use the GetAPIMessage() function in the runtime to obtain a standardised text line for any Notes API error code that is to be displayed or logged.

2.4 DXException Class

Header File: DXCommon/DXException.h

The DXException class is used by various parts of the DX runtime to pass exception details from the point of detection to the exception handling context. This is a standard “Frisbee” class i.e. it is thrown by the exception detection code and then caught by the appropriate exception handling context.

DX class implementations will never percolate a DXException across the API boundary exposed to an application, any exception that is detected internally will be handled internally and conerted to an appropriate application response according to the API contract.

2.5 ElapsedTimer Class

Header File: DXCommon/ElapsedTimer.h

Objects of the ElapsedTimer class provide a standard means of establishing elapsed (i.e. wall clock) time between events.

The class provides a getElapsed() method that returns a clock_t value containing the number of elapsed “ticks” since the object was created. A “tick” is platform dependent, the CLOCKS_PER_SEC defined symbol provides the number of “ticks” in a second on the target platform.

For longer intervals the class provides the getElapsedMillis() method that returns a clock_t value containing the number of elapsed milliseconds since the object was created.

The runtime provides a default ElapsedTimer object that is created during initialisation of the runtime, this object can be accessed through the “RunningTime” member of the runtime object.

2.6 RunSettings Class

Header File: DXCommon/RunSettings.h

An object of the RunSettings class or more usually a class that extends the RunSettings class is used to contain application and run specific information that conditions the configuration of the runtime and the application. Static application wide configuration data is set in the object along with parameters passed on the command line, these are then used to control the configuration of the runtime and the application.

As an example the name, version and short description of the application are used at various places in the runtime, these are accessed through the RunSettings object that is used to initialise the runtime. The runtime also has a special database called the “Repository” (this database is optional for the runtime) it can be used for different purposes such as the destination for persistent logging. Typically the server and file name for the repository would be supplied as command line parameters and then set in the RunSettings object, if set the repository database will be opened during initialisation of the runtime.

2.7 Helper Class

Header File: DXCommon/Debug/Helper.h

This class is only implemented for Windows Platforms.

The Helper class is used to establish an object that provides additional diagnostic capabilities to the runtime environment. Objects of this class should only be constructed in DEBUG build configurations of an application.

The class provides services for monitoring application memory usage and producing Core Dumps on demand.

2.8 CommandHandler Class

Header File: DXCommon/MTX/CommandHandler.h

An object of the CommandHandler class or more usually a class that extends it is constructed by the application and activated through a call to the runtime. The class provides default handling of the Message Queue (MQ) for Server Add-In tasks. Extending classes can implement additional commands and/or override, modify or extend the processing associated with the default command set.

2.9 TransactionHandler Class

Header File: DXCommon/MTX/TransactionHandler.h

This class must be extended to provide a default implementation for handling the reading, dispatch and status recording for a queue of transactions. A transaction handler object and the associated transaction queue would be created by the application and then dispatched for asynchronous execution. Once dispatched the object will monitor the associated transaction queue and respond by dispatching any transactions that appear in the queue. The handler recognises a structured set of sub-queues that allow for the automated retry of failed transactions and the repeated execution of transaction on a fixed time based schedule.

2.10 TransactionQueue Class

Header File: DXCommon/MTX/TransactionQueue.h

This class provides information that is used by the TransactionHandler to bind to the physical implementation of a transaction queue and determine a number of operational characteristics of the queue.

2.11 Runnable Class

Header File: DXCommon/Threads/Runnable.h

The runnable class defines an interface for any class that is to be asynchronously dispatchable by the multi-threaded runtime.

2.12 ThreadDispatcher Class

Header File: DXCommon/Threads/ThreadDispatcher.h

A singleton object of this class forms part of the multi-threaded runtime system, it has the responsibility for dispatching units of work that are ready to be executed into an available worker thread.

2.13 ThreadManager Class

Header File: DXCommon/Threads/ThreadManager.h

A singleton object of this class provides the master component of the multi-threaded runtime system .

2.14 ThreadManagerPolicy Class

Header File: DXCommon/Threads/ThreadManagerPolicy.h

An object of this class contains information used by the thread manager to configure the multi-threaded runtime system. An application can configure an object of this class and use it in the creation of the runtime system to influence many settings and constraints that are used by the runtime system.

2.15 ThreadMonitor Class

Header File: DXCommon/Threads/ThreadMonitor.h

A singleton object of this class forms part of the multi-threaded runtime system, it has the responsibility for monitoring several aspects of the multi-threaded runtime system this includes such housekeeping actions as writing log entries.

2.16 ThreadScheduler Class

Header File: DXCommon/Threads/ThreadScheduler.h

A singleton object of this class forms part of the multi-threaded runtime system, it has the responsibility for maintaining the relative priority of units of work that are waiting to be executed.

2.17 WorkerThread Class

Header File: DXCommon/Threads/WorkerThread.h

This class provides objects that manage the execution of units of work in a single thread.

3. Functional Classes

3.1 DXACLRuleSetParser Class

Header File: DXCommon/ACL/DXACLRuleSetParser.h

The DXACLRuleSetParser class is a factory class for DXACLRuleSet objects (see later) that can be used to manipulate the ACL of databases. The factory class will create an DXACLRuleSet object from an XML specification document (see Appendix A. for the specification of this document).

3.1.1 API

3.1.1.1 Constructor

```
DXACLRuleSetParser(ExecEnvironment *xeParent, int iThreadID)
```

Name	Type	Use
xeParent	ExecEnvironment *	Pointer to the current runtime object
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is creating the object.

NOTE: After construction the address of a DXResourceLoader object must be set in the rlCurrent member of the object.

3.1.1.2 parseTheseRules

```
DXACLRuleSet * parseTheseRules(char *szRuleSet, int iThreadID)
DXACLRuleSet * parseTheseRules(NOTEHANDLE hnSource, char *szItemName, int iThreadID)
```

Name	Type	Use
szRuleSet	char *	Pointer to a null terminated string that specifies the location of the XML document that contains the ACL rules. This string can be a URL or a file name.
hnSource	NOTEHANDLE	Handle of a Notes document that contains a field with the XML specifying the ACL rules.
szItemName	char *	Pointer to a null terminated string that contains the name of the field that contains the XML specifying the ACL rules.
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is creating the object.

This function will take an XML document containing ACL rules and will return a pointer to the DXACLRuleSet object constructed from the ACL rules. The function will return NULL if the XML document could not be found or was non-conforming.

NOTE: The address of a DXResourceLoader object must be set in the rlCurrent member of the object before these functions are invoked.

3.1.2 *Usage Notes*

The DXACLRuleSetParser object is thread safe, multiple threads can make parseTheseRules() function calls in parallel.

3.2 DXACLRuleSet Class

Header File: DXCommon/ACL/DXACLRuleSet.h

The DXACLRuleSet is a container for a set of ACL rules that are derived from an XML specification document. Objects of this class can be used to manipulate the ACL of a database so that they conform to the specification.

3.2.1 API

3.2.1.1 Constructor

```
DXACLRuleSet(ExecEnvironment *xeParent, int iThreadID)
```

Name	Type	Use
xeParent	ExecEnvironment *	Pointer to the current runtime object
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is creating the object.

Do not use the constructor for objects of this class use the DXACLRuleSetParser factory class to parse an XML document containing the ACL rules to construct the set. Use the constructor to create an empty rule set that will be populated programmatically or by using the clone() method.

3.2.1.2 validate

```
BOOL validate(int iThreadID)
```

Name	Type	Use
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is creating the object.

The validate() function is not normally used directly by applications it is invoked by the factory class after constructing a rule set from an XML document. If used in an application that manipulates the rules in a rule set after they have been loaded then invoke the validate() function to check the internal consistency of the rule set. The function returns TRUE if the rule set is internally consistent and FALSE if there is an inconsistency. If the function returns FALSE messages will be written to the log that indicate the cause of the inconsistency. Do not invoke any other functions on a rule set object that is in an inconsistent state.

3.2.1.3 coerce

```
BOOL coerce(HANDLE hACL, int iThreadID)
```

Name	Type	Use
hACL	HANDLE	Handle of a database ACL that will be changed so that it conforms to the rules specified by the rule set.
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is creating the object.

The coerce method takes the handle to the ACL of a database and forces it to conform to the rules contained in the rule set. Individual ACL entries will be added, removed or modified as appropriate. The method returns TRUE if the ACL was modified to make it conform and FALSE if the ACL was already conformant to the rule set. If the method returns TRUE then the ACL should be saved to make the changes permanent.

NOTE: The DXACLRuleSet object maintains stateful information during a coerce operation and is therefore not thread safe.

3.2.1.4 clone

```
void clone(DXACLRuleSet *arsClone, int iThreadID)
```

Name	Type	Use
arsClone	DXACLRuleSet *	Pointer to an empty rule set object that will contain the cloned rules.
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is creating the object.

The clone() method is used to populate an empty DXACLRuleSet object with the same rules that are contained within the current DXACLRuleSet object.

The cloning operation is used to produce duplicate rule sets that can be used to modify the ACL of multiple databases at the same time.

3.2.2 Usage Notes

The coerce() method starts by “binding” the ACLRuleSet to the passed ACL. This process involves translating Role names into a privilege mask. While analysing the ACL for conformance with the rule set any adjustments are posted in the rule set object and then applied once the analysis is complete. This means that it is essential to only use a rule set on a single database at any one time, use the clone() method to populate a new rule set with the same rules and use the clone to process an additional database.

3.3 DbCopier Class

Header File: DXCommon/DBC/DbCopier.h

The DbCopier class provides an engine that copies databases making replica or non-replica copies of a source database. There are many options that can be applied during the copy operation. The copy process is optimised for high throughput processing many parts of the copy process in parallel, the class can only be used in a multi-threaded environment. The class is completely thread safe allowing multiple copy operations to be carried out in parallel by the same DbCopier object. A copy operation can be initiated either synchronously or asynchronously.

3.3.1 API

3.3.1.1 Constructor

```
DbCopier(MTExecutive *xeParent, int iThreadID)
```

Name	Type	Use
xeParent	MTExecutive *	Pointer to the current multi-threaded runtime object
iThreadID	int	Specify the number of the thread that is creating the object.

3.3.1.2 CopyThisDb

```
BOOL CopyThisDb(CopyRequest *crRQ, int iThreadID)
```

Name	Type	Use
crRQ	CopyRequest *	Pointer to the CopyRequest object that contains all of the parameters that control the copy operation.
iThreadID	int	Specify the number of the thread that is invoking the method.

This method provides a means of synchronously initiating a copy operation on a DbCopier object. The call is blocking and will wait until the copy operation completes before returning to the caller. The method returns TRUE if the copy operation was successful and FALSE if the operation failed.

Refer to the “Supporting Classes” section of this document for information about the CopyRequest class to see the options that are available for the copy operation.

3.3.1.3 Asynchronous Copy Operations

To invoke a copy operation asynchronously on a DbCopier object follow the steps below.

1) Create a CopyRequest object and populate it with the parameters needed for the copy operation.
2) Create a PartCopyRequest object and set the address of the CopyRequest object in the “Parent” member and set the “CopyAction” member to “COPY_DATABASE”.

3) Use the “PostARequest” method on the runtime object to dispatch the “PartCopyRequest” message to the DbCopier.

3.3.1.4 ProxyLogMessage Method

```
BOOL virtual ProxyLogMessage(CopyRequest *crRQ, char *szMsg, int iThreadID)
```

Name	Type	Use
crRQ	CopyRequest *	Pointer to the CopyRequest object that contains all of the parameters that control the copy operation.
szMsg	char *	Pointer to a null terminated character string containing the message that is being issued by the DbCopier.
iThreadID	int	The number of the thread that is invoking the method.

The ProxyLogMessage() method can be overridden in inheriting classes to provide access to the messages that are issued by the DbCopier. Overriding methods should return TRUE if they completed their processing or FALSE if they encountered an error while trying to process the message.

3.3.2 Usage Notes

Examine the QCopy application to see how the DbCopier is implemented and used in a real implementation.

3.4 DbMover Class

Header File: DXCommon/DBC/DbMover.h

The DbMover engine was originally developed to solve the problem of redistributing notes databases across multiple storage volumes without shutting down the Domino server or limiting the use of databases while they are being moved between storage volumes. Having addressed the problem of storage redistribution it was noted that the DbMover engine had other capabilities such as creating media backups and deleting databases already plumbed in and so minor code changes turned it into a more generic Notes database storage management tool. The name DbMover was retained as storage redistribution remains the principal use of the utility class.

The DbMover engine recognises three primitive actions that can be combined in different ways to accomplish different tasks. The primitive actions are Copy (C) which makes an online copy of an open (transaction logged) database, Delete (D) which deletes a database while it is online and Link (L) which creates a database link in place of the original database pointing to a new storage location (which can be outside of scope of the Notes Data Directory).

3.4.1 API

3.4.1.1 Constructor

```
DbMover(MTExecutive *xeParent, int iThreadID)
```

Name	Type	Use
xeParent	MTExecutive *	Pointer to the current multi-threaded runtime object
iThreadID	int	Specify the number of the thread that is creating the object.

3.4.1.2 MoveThisDb Method

```
BOOL MoveThisDb(MoveRequest *mrRQ, int iThreadID)
```

Name	Type	Use
mrRQ	MoveRequest *	Pointer to the MoveRequest object that contains all of the parameters that control the move or other operation.
iThreadID	int	Specify the number of the thread that is invoking the method.

This method provides a means of synchronously initiating a move or other operation on a DbMover object. The call is blocking and will wait until the move or other operation completes before returning to the caller. The method returns TRUE if the move or other operation was successful and FALSE if the operation failed.

Refer to the “Supporting Classes” section of this document for information about the MoveRequest class to see the operations and options that are available for the with this call.

3.4.1.3 Asynchronous Mover Operations

To invoke a Mover operation asynchronously on a DbMover object follow the steps below.

1) Create a MoveRequest object and populate it with the parameters needed for the operation.
2) Create a PartMoveRequest object and set the address of the MoveRequest object in the “Parent” member and set the “MoveAction” member to “MOVE_DATABASE”.
3) Use the “PostARequest” method on the runtime object to dispatch the “PartMoveRequest” message to the DbMover.

3.4.1.4 ProxyLogMessage Method

```
BOOL virtual ProxyLogMessage(MoveRequest *mrRQ, char *szMsg, int iThreadID)
```

Name	Type	Use
mrRQ	MoveRequest *	Pointer to the MoveRequest object that contains all of the parameters that control the current operation.
szMsg	char *	Pointer to a null terminated character string containing the message that is being issued by the DbMover.
iThreadID	int	The number of the thread that is invoking the method.

The ProxyLogMessage() method can be overridden in inheriting classes to provide access to the messages that are issued by the DbMover. Overriding methods should return TRUE if they completed their processing or FALSE if they encountered an error while trying to process the message.

3.4.2 Usage Notes

Examine the QMove application to see how the DbMover is implemented and used in a real application setting.

3.5 DbModifier Class

Header File: DXCommon/DBP/DbModifier.h

The DbModifier class provides an engine for modifying various properties of a database container.

Refer to the ModRequest supporting class to see the properties that can be modified using this engine.

3.5.1 API

3.5.1.1 Constructor

```
DbModifier(MTExecutive *xeParent, int iThreadID)
```

Name	Type	Use
xeParent	MTExecutive *	Pointer to the current multi-threaded runtime object
iThreadID	int	Specify the number of the thread that is creating the object.

3.5.1.2 ModifyThisDb Method

```
BOOL ModifyThisDb(ModRequest *mrRQ, int iThreadID)
```

Name	Type	Use
mrRQ	ModRequest *	Pointer to the ModRequest object that contains all of the parameters that control the modify operation.
iThreadID	int	Specify the number of the thread that is invoking the method.

This method provides a means of synchronously initiating a modify operation on a DbModifier object. The call is blocking and will wait until the modify operation completes before returning to the caller. The method returns TRUE if the modify operation was successful and FALSE if the operation failed.

Refer to the “Supporting Classes” section of this document for information about the ModRequest class to see the operations and options that are available for the with this call.

3.5.1.3 Asynchronous Modifier Operations

To invoke a Modify operation asynchronously on a DbModifier object follow the steps below.

1) Create a ModRequest object and populate it with the parameters needed for the operation.
2) Use the “PostARequest” method on the runtime object to dispatch the “ModRequest” message to the DbModifier.

3.5.1.4 ProxyLogMessage Method

```
BOOL virtual ProxyLogMessage(ModRequest *mrRQ, char *szMsg, int iThreadID)
```

Name	Type	Use
mrRQ	ModRequest *	Pointer to the ModRequest object that contains all of the parameters that control the current operation.
szMsg	char *	Pointer to a null terminated character string containing the message that is being issued by the DbMover.
iThreadID	Int	The number of the thread that is invoking the method.

The ProxyLogMessage() method can be overridden in inheriting classes to provide access to the messages that are issued by the DbModifier. Overriding methods should return TRUE if they completed their processing or FALSE if they encountered an error while trying to process the message.

3.5.2 Usage Notes

Examine the QModify application to see how the DbModifier is implemented and used in a generic application setting.

3.6 Helper Class

Header File: DXCommon/Debug/Helper.h

The Helper class is only available on the Windows platform and only exposes functionality when the application is compiled with Debug settings (Define _DEBUG). The class exposes a number of convenience methods to assist with application debugging.

3.6.1 API

3.6.1.1 Constructor

```
Helper(ExecEnvironment *xeParent, int iThreadID);
```

Name	Type	Use
xeParent	ExecEnvironment *	Pointer to the current runtime object
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is creating the object.

The constructor for a Helper object will redirect all C Runtime debugging outputs to STDOUT, so it should be noted that when used assertion failures are reported on the standard output stream and not through the normal assertion failure dialog. The constructor will also record the current memory usage so that subsequent reports of memory usage can show deltas against the initial state.

3.6.1.2 ReportMemoryUsage Method

```
void ReportMemoryUsage(BOOL bReset, int iThreadID)
void ReportMemoryUsage(BOOL bReset, BOOL bEcho, int iThreadID)
```

Name	Type	Use
bReset	BOOL	Set to TRUE to cause the initial values of the memory statistics to be reset to the current values (after they are reported). Set to FALSE to continue reporting against the initial values recorded when the object was constructed.
bEcho	BOOL	Set to TRUE to cause the reports of memory usage to be echoed to the console.
iThreadID	int	The number of the thread that is invoking the method.

The ReportMemoryUsage method is used to take a snapshot of memory usage and report on the current values, the deltas since last reported and the deltas since first recorded.

Sample Output:

```
Current(3) Working Set size: 30704 Kb, +328 Kb since last measured, +4580 Kb since first measured, Peak: 30704 Kb.
Current(3) Paged Pool use: 1252 Kb, 0 Kb since last measured, 0 Kb since first measured, Peak: 1252 Kb.
Current(3) Non-Paged Pool use: 10 Kb, 0 Kb since last measured, 0 Kb since first measured, Peak: 10 Kb.
Current(3) Normal Objects on the Heap: 28 , 0  since last measured, +7  since first measured, Peak: 28.
Current(3) Normal Objects Allocation: 124 Kb, 0 Kb since last measured, +2 Kb since first measured, Peak: 124 Kb.
Current(3) Client Objects on the Heap: 0 , 0  since last measured, 0  since first measured, Peak: 0.
Current(3) Client Objects Allocation: 0 Kb, 0 Kb since last measured, 0 Kb since first measured, Peak: 0 Kb.
```

The report indicates the sequence number of memory reports “Current(3)” indicates the third time that the reporting method has been called. Each line of output references a different memory statistic and shows the current, delta since last reported, delta since first reported and the peak measurement of the particular statistic. The statistics of particular focus for programmers are the count and size of “Normal Objects” on the heap, steady increases in these values would indicate a leak of C++ objects from within the application.

It should be noted that Debug compilations of DX applications also enables the C runtime memory leak tracing protocols. At any point in a program a call can be made to the C Runtime “CheckMemoryLeaks()” method and this will report on each object that is allocated on the Heap giving the source file and line number where it was allocated as well as the size of the object. A call to CheckMemoryLeaks should be made immediately before an application terminates this will show any objects that remain allocated and provides an excellent means of detecting and fixing leaks caused by failing to delete C++ objects or failing to free memory allocations.

3.6.1.3 CreateMemoryDump Method

```
void CreateMemoryDump(int iThreadID)
```

Name	Type	Use
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is creating the object.

This method can be called at any point in an application to create a “minidump” file of the application process including heap memory. The memory dump files are created in the “IBM_TECHNICAL_SUPPORT” sub-directory in the Notes data directory. The minidump files can be loaded into Visual Studio for contextual analysis or in the Windows Debugger (WinDbg).

The dump files are created with a standard name format:

DXDump-<appname>YYYYMMDD-HHMMSS.dmp.

Where <appname> is the name of the application and YYYYMMDD-HHMMSS is the timestamp that the dump was created.

3.6.2 Usage Notes

The address of the Debug Helper object can be inserted into certain DX Kernel objects in order to provide additional debugging capability.

Setting the address of the Helper in the CommandHandler->DebugHelper member enables the following additional commands.

MEMORY | MEM – Display the current memory statistics.

DUMP – Create a minidump (with heap memory).

Setting the address of the Helper in the TransactionHandler->DebugHelper member causes the transaction handler to report on the memory statistics of the application after completion of each transaction that is dispatched.

3.7 DesignManager Class

Header File: DXCommon/DM/DesignManager.h

The DesignManager class provides an engine that allows an application to add design elements to a Notes database from a collection of DXL sources.

3.7.1 API

3.7.1.1 Constructor

```
DesignManager(ExecEnvironment *xeParent)
```

Name	Type	Use
xeParent	ExecEnvironment *	Pointer to the current runtime object

3.7.1.2 LoadManifest Method

```
BOOL LoadManifest(char *szManifestID)
```

Name	Type	Use
szManifestID	char *	Pointer to a null terminated character string that contains the location of the manifest file (XML document) that defines all of the design elements that can be applied to the database. The location is normally expressed as an HTTP URL but can also be a file name or a program resource identifier in the form “#nnn”.

A design manifest is an XML document that contains a number of individual “Design Sets” and each Design Set can contain a number of “Design Elements”. Calls to LoadManifiest will cause the XML document to be located loaded into memory and parsed into a series of objects that can be used to apply collections of design elements to a database. Appendix B contains a specification for the XML document that is used to form a valid Manifest. Databases that are needed by DX utilities are all available on the Internet as “Virtual Templates”, a virtual template is a Manifest document and the sub-ordinate design elements that can be applied to a blank database to instantiate the complete design using a DesignManager instance. Refer to the RDBCreate application to see how this is implemented.

The method returns TRUE if the specified manifest was successfully loaded and parsed or FALSE if it was not.

3.7.1.3 DestroyManifest Method

```
void DestroyManifest(void)
```

A call to the DestroyManifest method will cause the currently loaded manifest and all of the objects that it represents to be deleted. A call to DestroyManifest should always be made after all the required design elements from a manifest have been applied to a database.

3.7.1.4 ApplyAllDesignSets Method

```
BOOL ApplyAllDesignSets(DBHANDLE hDB)
BOOL ApplyAllDesignSets(void)
```

Name	Type	Use
hDB	DBHANDLE	Handle of the database to which all of the design elements in a manifest will be applied.

A call to the ApplyAllDesignSets method can only be made on a DesignManager that has a valid loaded manifest. The call will apply all of the design elements listed in every Design Set in the Manifest.

If the form of the call that does not specify a database handle is used then the design elements will all be applied to the current Runtime Repository database.

The call returns TRUE if all design elements were applied and FALSE if not.

3.7.1.5 ApplyDesignSet Method

```
BOOL ApplyDesignSet(char *szSetName, DBHANDLE hDB)
BOOL ApplyDesignSet(char *szSetName)
```

Name	Type	Use
szSetName	char *	Pointer to a null terminated string containing the name of a design set in the manifest that will be applied to the database.
hDB	DBHANDLE	Handle of the database to which all of the design elements in the named design set will be applied.

A call to the ApplyDesignSets method can only be made on a DesignManager that has a valid loaded manifest. The call will apply all of the design elements listed in the specified Design Set in the Manifest.

If the form of the call that does not specify a database handle is used then the design elements will all be applied to the current Runtime Repository database.

The call returns TRUE if all design elements were applied and FALSE if not.

3.7.1.6 GetBuildInfo_s Method

```
void GetBuildInfo_s(char *szBuild, int iSize)
```

Name	Type	Use
szBuild	char *	Pointer to a buffer where a string will be returned that identifies the build of the DesignManager.
iSize	int	Size of the buffer.

This method will return a string that identifies the current build information of the Design Manager in the passed buffer.

3.7.1.7 GetDefaultDbTitle_s Method

```
void GetDefaultDbTitle_s(char *szTitle, int iSize)
```

Name	Type	Use
szTitle	char *	Pointer to a buffer where a string will be returned that contains a default database title derived from the current manifest.
iSize	int	Size of the buffer.

This method will return a string containing a default title that can be used for a new database, the title string is derived from the content of the manifest document that was applied to the database.

3.7.1.8 ValidateDesign Method

```
BOOL ValidateDesign(DBHANDLE hDB)
```

Name	Type	Use
hDB	DBHANDLE	Handle of the database which will have its design validated and if necessary fixed.

The ValidateDesign method can be used to check the design of a database that has been loaded from a manifest for logical consistency and apply limited corrections to make the design valid.

3.7.1.9 Control Modifiers

The following members may be set to specific values to modify the behaviour of the DesignManager engine.

`int` **`iDMDesignImportOption`** – Set this member to any of the DXLIMPORTOPTION_ symbolic values that can condition the processing in the DXLImporter, see the Notes API documentation for options.

The default is set to DXLIMPORTOPTION_REPLACE_ELSE_CREATE.

`BOOL bDMReplaceDbProperties` – Set this member to FALSE if you do not want the design to modify any DB properties.

The default is set to TRUE.

`BOOL bDMReplicaRequiredForReplaceOrUpdate` – Set this member to FALSE if you wish design updates or replace operations to be made using DXL that does not specify the Replica ID of the target database.

The default is set to TRUE;

`int iDMInputValidationOption` – Set this member to any of the Xml_Validate values that can be set in the DXLImporter.

The default is set to Xml_Validate_Never.

`BOOL bDMExitOnFirstFatalError` – Set this member to FALSE to prevent the DXLImporter from abandoning processing of an XML stream when it first encounters an error.

`int iDMUnknownTokenLogOption` – Set this member to any of the DXLLOGOPTION_ symbolic values to determine the logging behaviour of the DXLImporter with unknown XML tokens.

3.7.2 Usage Notes

3.8 DXResourceLoader Class

Header File: DXCommon/DXResourceLoader.h

The DXResourceLoader class provides an engine for loading a resource (usually an XML document) into memory. The resource can be loaded from a remote HTTP source, a native file, a resource embedded in the application (windows only), a TEXT or TEXT_LIST item in a Notes Document or a Rich Text item in a notes document. The engine is capable of returning the resource loaded completely in-memory or providing the resource on-demand to a reader function such as the XML_READ_FUNCTION of a DXLImporter. The class also provides the capability for Posting a resource file into a new Notes Document via HTTP.

3.8.1 API

3.8.1.1 Constructor

```
DXResourceLoader(ExecEnvironment *xeParent, int iThreadID)
```

Name	Type	Use
xeParent	ExecEnvironment *	Pointer to the current runtime object
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the

		thread that is creating the object.

3.8.1.2 loadThisResource Method

```
MEMHANDLE loadThisResource(char *szResourceSource, int iThreadID)
MEMHANDLE loadThisResource(NOTEHANDLE hnSource, char *szItemName, int iThreadID)
```

Name	Type	Use
szResourceSource	char *	Pointer to a null terminated character string that contains the location of the resource file (XML document) that is to be loaded. The location is normally expressed as an HTTP URL but can also be a file name or a program resource identifier in the form “#nnn”.
hnSource	NOTEHANDLE	Handle of a notes document that contains an item that is carrying the requested resource.
szItemName	char *	Pointer to a null terminated string that contains the name of the item on a notes document that contains the requested resource.
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is creating the object.

The loadThisResource method does exactly what it says on the box, it will take the passed parameters that identify the location of the resource and will load it into memory and return a handle to the memory that contains the loaded resource. If the resource cannot be loaded then the method returns NULLHANDLE.

The location of the resource can be specified in any of the following ways.

A HTTP URL – e.g. http://some.domain.com/arbitrary/location/specifier.

A File Name – e.g. “c:\folder\subforlder\resource.xxx” or “/home/folder/subforder/resource.xxx”.

A Resource ID – e.g. “#0251” (windows only).

An Item on a Note – specified as the handle to an open note and the name of the item that contains the resource.

3.8.1.3 loadThisResourceOnDemand Method

```
DXRESOURCEHANDLE loadThisResourceOnDemand(char *szResourceSource, int iThreadID)
DXRESOURCEHANDLE loadThisResourceOnDemand(NOTEHANDLE hnSource, char *szItemName, int
iThreadID)
DXRESOURCEHANDLE loadThisResourceOnDemand(MEMHANDLE hResource, int iThreadID)
```

Name	Type	Use
szResourceSource	char *	Pointer to a null terminated character string that contains the location of the resource file (XML document) that is to

		be loaded. The location is normally expressed as an HTTP URL but can also be a file name or a program resource identifier in the form "#nnn".
hnSource	NOTEHANDLE	Handle of a notes document that contains an item that is carrying the requested resource.
szItemName	char *	Pointer to a null terminated string that contains the name of the item on a notes document that contains the requested resource.
hResource	MEMHANDLE	Handle of a resource that has already been loaded into memory.
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is creating the object.

The loadThisResourceOnDemand method does exactly the same as the loadThisResource method except the resource is not loaded immediately. The resource is loaded when the XML reader function is invoked. The interface follows the protocols for the XML reader defined for the DXLImporter, it will continue to return partial results in the passed buffer until the end of the resource stream is reached whereupon it will return 0 bytes.

The location of the resource can be specified in any of the following ways.

A HTTP URL – e.g. http://some.domain.com/arbitrary/location/specifier.

A File Name – e.g. "c:\folder\subforlder\resource.xxx" or "/home/folder/subforder/resource.xxx".

A Resource ID – e.g. "#0251" (windows only).

An Item on a Note – specified as the handle to an open note and the name of the item that contains the resource.

Memory Handle – The handle of a resource that has already been loaded into memory.

Calls to loadThisResourceOnDemand will return a DXRESOURCEHANDLE or NULLHANDLE if the resource specifiers were invalid.

The DXRESOURCEHANDLE must be passed as the context parameter to the XML reader routine via whatever process will invoke the reader.

A DXRESOURCEHANDLE must be destroyed after use by passing it in a call to the disposeResourceHandle method.

3.8.1.4 cloneResourceHandle Method

```
DXRESOURCEHANDLE cloneResourceHandle(DXRESOURCEHANDLE hdrPrime, int iThreadID)
```

Name	Type	Use
hdrPrime	DXRESOURCEHANDLE	Handle of an existing load on demand resource.

The resource that underlies a DXRESOURCEHANDLE contains stateful information and therefore can only be used once. The cloneResourceHandle method provides the capability to provide copies on an on-demand resource to be consumed by multiple readers.

3.8.1.5 disposeResourceHandle Method

```
void disposeResourceHandle(DXRESOURCEHANDLE hdrUsed, int iThreadID)
```

Name	Type	Use
hdrUsed	DXRESOURCEHANDLE	Handle of an existing load on demand resource.

Call the disposeResourceHandle method to destroy an on demand resource handle and free the underlying objects and memory associated with the resource.

3.8.1.6 WrapperXMLReader Method

```
static DWORD LNCALLBACK WrapperXMLReader(unsigned char *ucBuffer, DWORD dwMaxToRead,
void *pContext)
```

The signature of this method conforms to the Notes API XML_READ_FUNCTION signature and therefore can be passed into any call in the Notes API that needs to load XML, such as the DXLImport method.

Name	Type	Use
ucBuffer	UCHAR *	Pointer to a buffer where the next chunk of XML will be returned by the routine.
dwMaxToRead	DWORD	The maximum number of characters that can be read into the passed buffer.
pContext	void *	The DXRESOURCEHANDLE of the resource that is to be read on behalf of the caller.

The method will return the number of characters that have been returned in the buffer. The method should be called continuously until it returns 0 indicating that the complete resource has been read.

3.8.1.7 uploadThisResource Method

```
MEMHANDLE uploadThisResource(char *szFileName, int iFileType, DXUploadContext
*ducThis, int iThreadID)
```

```
MEMHANDLE uploadThisResource(MEMHANDLE hmResource, int iFileType, DXUploadContext
*ducThis, int iThreadID)
```

Name	Type	Use
szFileName	char *	Pointer to a null terminated character string that contains the file name of the resource that is to be uploaded.
hmResource	MEMHANDLE	Handle of an area of memory that contains the resource that is to be uploaded.
iFileType	Int	An integer value that indicates the type of resource file that is being uploaded. Use one of the following symbolic values.

		UPLOAD_FILE_TEXT – for plain text files UPLOAD_FILE_XML – for XML documents UPLOAD_FILE_BINARY – for resources with binary content
ducThis	DXUploadContext *	Pointer to a DXUploadContext object that has been populated to describe the target web page where the resource will be uploaded.
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is creating the object.

The method returns a MEMHANDLE containing the response HTML from the posting operation, it is the responsibility of the caller to parse the response to determine the remote success or otherwise. The method will return NULLHANDLE if the local processing fails.

3.8.2 Usage Notes

Refer to the use of this class in the DesignManager class and the implementation in the RDBCreate application to see examples of immediate and on demand loading and consumption of resources.

3.9 DominoExplorer Class

Header File: DXCommon/DX/DominoExplorer.h (Single Threaded)

Header File: DXCommon/MTDX/DominoExplorer.h (Multi Threaded)

The DominoExplorer (Domino eXplorer) class is the original functional class from which the complete DX package takes its name. The class provides an engine to resolve an arbitrary scope within a Domino infrastructure and return each of the databases within the scope to a callout function.

The DominoExplorer class is implemented as either a single threaded class or a multi threaded class

3.9.1 API

3.9.1.1 Constructor

```
DominoExplorer(MTExecutive *xeParent, int iInstance, int iThreadID)
DominoExplorer(ExecEnvironment *xeParent, int iThreadID)
```

Name	Type	Use
xeParent	MTExecutive *	Pointer to the current multi threaded runtime object
xeParent	ExecEnvironment *	Pointer to the current single threaded runtime object
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is creating the object.

Include the appropriate header file for the DominoExplorer class in your application and construct the explorer object pass it a pointer to the appropriate runtime.

3.9.1.2 ProcessReqest Method

```
BOOL ProcessRequest(DXRequest *dxrRQ, int iThreadID)
```

Name	Type	Use
dxrRQ	DXRequest*	Pointer to a DXRequest object that specifies the desired scope and processing exits and options.
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is creating the object.

Refer to the DXRequest class in the Supporting Classes section of this document for details of how to set up the request for processing by the Domino Explorer.

The method will return TRUE if all databases within the specified scope were successfully processed and FALSE if not.

3.9.1.3 Asynchronous Processing

To invoke an explorer operation asynchronously on a DominoExplorer object follow the steps below.

1) Create a DXRequest object and populate it with the parameters needed for the operation.
2) Use the “PostARequest” method on the runtime object to dispatch the “DXRequest” message to the DominoExplorer.

3.9.2 Usage Notes

Refer to the QCopy application and the processing of “Feeder Transactions” to see the implementation of the Domino Explorer in a real application setting.

3.10 DXNoteScanner Class

Header File: DXCommon/DX/DXNoteScanner.h (Single Threaded)

Header File: DXCommon/MTDX/DXNoteScanner.h (Multi Threaded)

The DXNoteScanner class provides a sophisticated selection engine that can derive multiple collections of notes from a Notes database based on caller specified criteria.

3.10.1 API

3.10.1.1 Constructor

```
DXNoteScanner(MTExecutive *xeParent, int iThreadID)
```

Name	Type	Use
xeParent	MTExecutive *	Pointer to the current multi threaded runtime object
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is creating the object.

3.10.1.2 ScanDb Method

```
BOOL ScanDb(DXNoteScanReq *nsrCurrent, int iThreadID)
```

Name	Type	Use
nsrCurrent	DXNoteScanReq *	Pointer to the scan request object that defines the collections that are to be built.
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is creating the object.

Refer to the definition of the DXNoteScanReq class to see the options and limitations for searching. The method returns TRUE if the search(es) were completed and FLASE if they could not be completed. The usual cause of a FALSE return is that invalid search parameters were passed in the scan request.

3.10.2 Usage Notes

3.11 ObjectCache Class

Header File: DXCommon/Misc/ObjectCache.h

The ObjectCache class provides the partial implementation of objects that perform write through cache processing for a collection of arbitrary objects. The class is partially abstract some methods must be implemented in an inheriting class. The cache mechanism is implemented in the base class, methods that need to be aware of the type of object being cached need to be implemented in an inheriting class.

3.11.1 API

3.11.1.1 Constructor

```
ObjectCache(void)
```

Default constructor for the base class.

3.11.1.2 initCache Method

```
void initCache(int iMaxEntries, int iStrategy, int iThreadID)
```

Name	Type	Use
iMaxEntries	int	The maximum number of entries that will be held in the cache memory.
iStrategy	int	An integer value that defines the strategy that will be followed by the cache. Use one of the following symbolic values. `CACHE_STRATEGY_MFU` – Most Frequently Used `CACHE_STRATEGY_MRU` – Most Recently Used
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is creating the object.

Inheriting classes MUST invoke this method to initialise the cache structure and set the constraints for cache operation.

Specify `CACHE_STRATEGY_MFU` to cause the cache to favour the most frequently referenced objects. If the cache becomes full then the object with the lowest reference count will be dismissed from the cache.

Specify `CACHE_STRATEGY_MRU` to cause the cache to favour the most recently referenced objects. If the cache becomes full then the object that has not been referenced for the longest time will be dismissed from the cache.

3.11.1.3 getCachedObject Method

```
void *getCachedObject(char *szKey, int iThreadID)
```

Name	Type	Use
szKey	char *	A pointer to a null terminated character string that contains the key of the object that is to be retrieved from the cache.
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is making the call.

The contract for this method is that it will always return a pointer to a valid object of the type that is being cached. If the object with the matching key is already in the cache then the pointer to that object is returned to the caller. If the object is not currently in the cache then the createObject method is invoked (this method must be instantiated in the inheriting class) to create a new object, either from a backing store or a brand new initialised object, the new object is added to the cache and returned to the caller.

3.11.1.4 getCachedObjectExt Method

```
void *getCachedObjectExt(char *szKey, BOOL bCreate, int iThreadID)
```

Name	Type	Use

szKey	char *	A pointer to a null terminated character string that contains the key of the object that is to be retrieved from the cache.
bCreate	BOOL	Set this to TRUE if the method will follow the same contract as the getCachedObject method. Specify FALSE if you want the method to return NULL if the requested object is not present in the cache.
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is making the call.

The contract for this method depends on the value passed in the bCreate parameter. If TRUE then the contract is that it will always return a pointer to a valid object of the type that is being cached. If the object with the matching key is already in the cache then the pointer to that object is returned to the caller. If the object is not currently in the cache then the createObject method is invoked (this method must be instantiated in the inheriting class) to create a new object, either from a backing store or a brand new initialised object, the new object is added to the cache and returned to the caller. If FALSE is specified then the method will return the pointer to the object that matches the passed key. If the requested object is not currently present in the cache then the method will return NULL.

3.11.1.5 getNextCachedObject Method

```
void *getNextCachedObject(void *pCurrentObject, int iThreadID)
```

Name	Type	Use
pCurrentObject	void *	A pointer to the last object retrieved from the cache or specify NULL to get the first object.
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is making the call.

This method provides an iterator over the current contents of the cache. Specifying NULL for the current object parameter will return a pointer to the first object in the cache or NULL if the cache is empty. The method will return NULL if a pointer to the last object in the cache is passed.

3.11.1.6 Drain Method

```
void drain(int iThreadID)
```

Name	Type	Use
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is making the call.

This method should be called when the cache is about to be destroyed, this causes all of the objects that remain in the cache to be passed to the cacheOut interface. The cacheOut interface can be used to optionally serialize the objects to a backing store at a minimum the underlying object must be destroyed by the interface.

3.11.1.7 getEntries Method

```
DWORD getEntries(void)
```

This method returns a DWORD containing the count of objects currently in the cache.

3.11.1.8 dump Method

```
void dump(int iThreadID)
```

Name	Type	Use
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is creating the object.

This method performs the same function as the drain method but passes a flag to the cacheOut interface to indicate that objects should not be serialised to any backing store.

3.11.1.9 selectObject Method

```
virtual BOOL selectObject(void *pObject, char *szKey, int iThreadID)
```

Name	Type	Use
pCurrentObject	void *	A pointer to an object currently in the cache.
szKey	char *	A pointer to a null terminated character string that contains the key of the object that is to be retrieved from the cache.
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is making the call.

This method must be implemented in an inheriting class. The method must test if the object that is passed matches the key that is passed. If the object matches the key then the method must return TRUE otherwise it should return FALSE.

3.11.1.10 createObject Method

```
virtual void *createObject(char *szKey, int iThreadID)
```

Name	Type	Use
szKey	char *	A pointer to a null terminated character string that contains the key of the object that is to be created.

iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is making the call.

This method must be implemented in an inheriting class. The method must construct a new object based on the passed key and return a pointer to that object. The method can create the object by loading the serialised form from a backing store or simply by instantiating and populating the object.

3.11.1.11 cacheOut Method

```
virtual void cacheOut(void *pObject, BOOL bWrite, int iThreadID)
```

Name	Type	Use
pObject	void *	A pointer to an object currently in the cache.
bWrite	BOOL	If set to TRUE then the passed object should be serialized to a backing store and then destroyed. If set to FALSE then the passed object should just be destroyed.
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is making the call.

This method must be implemented in an inheriting class. The primary responsibility of the method is to destroy the passed object as it is removed from the cache. Implementations may optionally take notice of the bWrite parameter and serialise the object to a backing store if requested before destroying the object.

3.11.1.12 lockTheCache Method

```
void lockTheCache(int iThreadID)
```

Name	Type	Use
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is making the call.

The lockTheCache method will assert a mutex over the cache on behalf of the caller. If the mutex is already asserted by another thread then the call will block until the mutex becomes free and can be asserted. If the caller already owns the mutex then the call will just return.

3.11.1.13 unlockTheCache Method

```
void unlockTheCache(int iThreadID)
```

Name	Type	Use
iThreadID	int	For single threaded applications always specify 0 to indicate

		the main thread of a program otherwise specify the number of the thread that is making the call.

The unlockTheCache method will release the mutex over the cache on behalf of the caller. If the mutex is not held by the caller then the call just returns.

3.11.1.14 Statistics Members

The following public members are exposed to allow callers to collect statistical data on the performance of the cache.

- DWORD dwSearches – count of the number of searches performed on the cache.
- DWORD dwSearched – count of objects that were examined during searches.
- DWORD dwNoHits – count of searches that failed to locate the requested object.
- DWORD dwReorgs – count of cache reorganisations that were attempted.
- DWORD dwWrites – count of the number of objects that were removed from the cache.
- DWORD dwWaits – count of the number of calls to lock the cache that had to wait.
- DWORD dwWaitQuanta – count of wait cycles while waiting for the cache mutex.
- DWORD dwRSkips – count of cache reorganisations that were abandoned.
- DWORD dwMFUIProms – count of objects that were promoted in the cache because the reference count exceeded that of other objects.
- DWORD dwMFUIPQs – count of slots over which objects were promoted in the cache.

3.11.2 Usage Notes

3.12 TimeBasedArray Class

Header File: DXCommon/Misc/TimeBasedArray.h

The TimeBasedArray class provides the functionality to manage arrays of counters or accumulators against a dynamic time base. Entries are incremented or accumulated against points in time and the class will dynamically manage recording the values against a range of time buckets. The class also provides the capability to serialise the contents of the arrays to a Notes document and marshall the arrays from a Notes document.

By default variables are managed against the following time buckets, the buckets are measured from the latest timestamp recorded.

1. Hourly – for the latest 7 days.
2. Daily – for the latest 64 days.

3. Weekly – for the latest 24 weeks.
4. Monthly – for the latest 24 months.
5. Yearly – for the latest 10 years.
6. Grand Total.

3.12.1 API

3.12.1.1 Constructor

```
TimeBasedArray()
TimeBasedArray(int iVars, BOOL bRecordHourly)
```

Name	Type	Use
iVars	int	The number of independent variables that will be recorded in the array.
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is creating the object.

The TimeBasedArray is normally implemented in an inheriting class so the default constructor will be used. If the TimeBasedArray does not need to be serialised to a Notes document then the class can be instantiated natively. Inheriting classes must perform the following actions (usually in their constructors) before using the array.

1. Set the VarCount member (int) to the number of independent variables that are to be handled by the array object.
2. Set the Hourly member (BOOL) to TRUE if hourly buckets will be recorded and FALSE if not.
3. Set the tdAnchor member (TIMEDATE) to the TIMEDATE_MINIMUM constant value.
4. Call the initArray() method to initialise the structures underlying the array.

3.12.1.2 isArrayInitialised Method

```
BOOL isArrayInitialised(void)
```

This method returns TRUE if the array has been correctly initialised and FALSE if not.

3.12.1.3 marshall Method

```
BOOL marshall(char *szLatestTDFieldName, NOTEHANDLE hnSNote)
```

Name	**Type**	**Use**
szLatestTDFieldName	char *	A pointer to a null terminated character string that contains the name of an item on the notes document that contains the latest timestamp recorded.

hnSNote	NOTEHANDLE	Handle of the note that contains the serialized array data.

This method will call the getFieldName method to get the base field name for each variable recorded in the array. For each variable the method will load values from 4 or 5 separate number array items and one number item to populate the array. The item names for the arrays are constructed using the base name for the variable and a suffix of “Hourly” (Optional), “Daily”, “Weekly”, “Monthly” and “Yearly” the number field will use the suffix “Total”.

3.12.1.4 serialize Method

```
BOOL serialize(char *szLatestTDFieldName, NOTEHANDLE hnSNote)
```

Name	Type	Use
szLatestTDFieldName	char *	A pointer to a null terminated character string that contains the name of an item on the notes document to contain the latest timestamp recorded.
hnSNote	NOTEHANDLE	Handle of the note that will contain the serialized array data.

This method is the counterpart of the marshall method. It will serialise the information contained in the TimeBasedArray to a Notes document. The method will call the getFieldName method to get the base field name for each variable to be recorded in the array. For each variable the method will store values in 4 or 5 separate number array items and one number item from the values recorded in the array. The item names for the arrays are constructed using the base name for the variable and a suffix of “Hourly” (Optional), “Daily”, “Weekly”, “Monthly” and “Yearly” the number field will use the suffix “Total”.

3.12.1.5 increment Method

```
void increment(int iVarIndex, TIMEDATE *tdEvent)
```

Name	Type	Use
iVarIndex	int	The index number of the variable to be incremented, numbered from 1 upwards.
tdEvent	TIMEDATE	The timestamp associated with the count increment.

The increment method will increment the counters in the TimeBasedArray that apply to the specified timestamp and variable. If the timestamp is later than the previous latest timestamp recorded then the array will reorganise itself to reflect the new timebase.

3.12.1.6 add Method

```
void add(int iVarIndex, TIMEDATE *tdEvent, DWORD dwValue)
```

Name	Type	Use
iVarIndex	int	The index number of the variable to be incremented, numbered from 1 upwards.
tdEvent	TIMEDATE	The timestamp associated with the count increment.
dwValue	DWORD	The value by which the count will be incremented

The add method will increment the counters in the TimeBasedArray by a specified amount that apply to the specified timestamp and variable. If the timestamp is later than the previous latest timestamp recorded then the array will reorganise itself to reflect the new timebase.

3.12.1.7 decrement Method

```
void decrement(int iVarIndex, TIMEDATE *tdEvent)
```

Name	Type	Use
iVarIndex	int	The index number of the variable to be decremented, numbered from 1 upwards.
tdEvent	TIMEDATE	The timestamp associated with the count decrement.

The decrement method will decrement the counters in the TimeBasedArray that apply to the specified timestamp and variable. If the timestamp is later than the previous latest timestamp recorded then the array will reorganise itself to reflect the new timebase.

3.12.1.8 subtract Method

```
void subtract(int iVarIndex, TIMEDATE *tdEvent, DWORD dwValue)
```

Name	Type	Use
iVarIndex	int	The index number of the variable to be decremented, numbered from 1 upwards.
tdEvent	TIMEDATE	The timestamp associated with the count decrement.
dwValue	DWORD	The value by which the count will be decremented

The subtract method will decrement the counters in the TimeBasedArray by a specified amount that apply to the specified timestamp and variable. If the timestamp is later than the previous latest timestamp recorded then the array will reorganise itself to reflect the new timebase.

3.12.1.9 getFieldName Method

```
void virtual getFieldName(int iVarNum, char *Bfr)
```

Name	Type	Use
iVarIndex	int	The index number of the variable to be decremented, numbered from 1 upwards.
Bfr	char *	A pointer to a buffer that will be populated with the base filed name associated with the variable identified by the index number.

This method should be implemented in inheriting classes. The value set in the buffer should be a null terminated character string and should not exceed `TBA_MAX_BASENAME` characters in length.

There is a default implementation of this method that returns "Counter<n>" where <n> is the index number.

3.12.1.10 getArraySize Method

```
int getArraySize(int iTimeIndex)
```

Name	Type	Use
iTimeIndex	int	A value that indicates which timebase array size should be returned, use one of the following symbolic values. `TBA_HOURLY_INDEX` – Hourly Array `TBA_DAILY_INDEX` – Daily Array `TBA_WEEKLY_INDEX` – Weekly Array `TBA_MONTHLY_INDEX` – Monthly Array `TBA_YEARLY_INDEX` – Yearly Array `TBA_TOTAL_INDEX` – Total Array

Calls to this method will return the size of a particular timebase area of the array.

3.12.1.11 getCounterValue Method

```
DWORD getCounterValue(int iVarIndex, int iTimeIndex, int iElement)
```

Name	Type	Use
iVarIndex	int	The index number of the variable to be returned, numbered from 1 upwards.
iTimeIndex	int	A value that indicates which timebase array size should be returned, use one of the following symbolic values. `TBA_HOURLY_INDEX` – Hourly Array `TBA_DAILY_INDEX` – Daily Array `TBA_WEEKLY_INDEX` – Weekly Array `TBA_MONTHLY_INDEX` – Monthly Array `TBA_YEARLY_INDEX` – Yearly Array `TBA_TOTAL_INDEX` – Total Array

iElement	int	The index of the array cell to be returned, zero based maximum value can be array size – 1.

Calls to this method will return a DWORD value from a particular cell in the array. The target cell is identified by the variable index number, the timebase index number and the element index of the sub-array.

3.12.2 Usage Notes

The class provides a simple means of maintaining running recordings of activities against time. Multiple variables are provided in the class so that they can all be maintained against a common timebase.

3.12.2.1 UI Implementation

It is possible to implement a Form to display multiple variables using only a single sub-form that displays the serialised form of the TimeBasedArray for a single variable, this can greatly reduce the development time for displaying data captured through this mechanism.

1) The Problem

Refer to the Serialize method to see how data is stored in a document during a serialize() function. You end up with a series of lists with field names that are a concatenation of a variable part and a fixed part. If you want to display multiple Time Based Arrays in a Form you really do not want to have to hand code all of the code for each variable to get them into the UI.
What would be nice would be to define a few fields and then just include a sub-form (the same one) for each variable and the sub-form dynamically computes what to display.

2) The IDEA

Define a few hidden fields at the top of the form as follows

FieldPrefix := "VarName1":"VarName2":"VarName3" ......   The names of the variables i.e. fixed parts of the field names
FieldDescription := "Variable #1":"Variable #2":"Variable #3"   The descriptive names for the variables
FieldLatestStamp := "FieldName"   The name of the field that contains the latest timestamp
FieldIndex := 0   Index of the field you are currently displaying

The fill the Form as follows :-

<Variable Display Subform>   For variable #1
<hidden field - increment FieldIndex>
<Variable Display Subform>   For variable #2
<hidden field - increment FieldIndex>

NOTE: The subform must use computed text rather than computed for display fields to display the data to prevent duplicate field names.

3) The Implementation

a) Problem - when you try to insert the second instance of the sub-form then designer says "NO – sub-form is already in the form".

b) Try computing the sub-form name for the second and subsequent sub-forms - ok in designer then client says - "NO – sub-form is already in the form".

c) Give a sequential number of aliases to the sub-form and use those names in the computed sub-form formula - and hey the man from Lotus he say "YES" - cool.

So now one can expend energy once to make a single good sub-form and then just include it as many times as you like (subject to number of aliases you can define).

I do love it when a bit of lateral thinking contributes positively to the inherent laziness of developers.

4) Formula for Computed Text to Display a Cell Value

The Hidden fields at the top of the form as set as follows.

FieldPrefix := "Created":"Modified"
FieldLatestStamp := "LatestTimeStamp"
FieldDescription := "Creation Times":"Modification Times"
FieldIndex := 0

The subform shows 3 fields all computed text calculated as follows.

The caption: @Return(@Subset(@Subset(FieldDescription; FieldIndex + 1); -1))

Latest TimeStamp: @Return(@Text(@GetField(FieldLatestStamp);"D0T0Z2"))

This Month: @Return(@Text(@Subset(@GetField(@Subset(@Subset(FieldPrefix; FieldIndex + 1) ; -1) + "Month"); 1); "F0"))

The computed for display field between the two subforms has the following value:

@SetField("FIeldIndex"; FieldIndex + 1); @Return("This would not be displayed - " + @Text(FieldIndex; "F0"))

3.13 AutoScaleFrequencyAnalyser Class

Header File: DXCommon/Misc/AutoScaleFrequencyAnalyser.h

The AutoScaleFrequencyAnalyser call provides a means of recording frequency counts against a variable data value without having to know the bounds of the variable data value before you start. The class provides a dynamically scaled but fixed size array that dynamically rescales itself to accommodate changing data bounds.

3.13.1 API

3.13.1.1 Constructor

```
AutoScaleFrequencyAnalyser(int Buckets, int Outliers, int RenormBkts)
```

Name	Type	Use

iBuckets	int	The number of intervals that will be recorded (array size)
iOutliers	int	The number of outlier data values that will be recorded, outliers are recorded at the high end and the low end of the data bounds and do not cause the array to be re-scaled.
iRenormBkts	int	The number of data intervals that will be used to extend the data bounds to incorporate close outliers at the end of recording.

As a guide set the number of outliers to 10% of the number of data intervals (buckets). Set the iRenormBkts to 5% of the number of the number of data intervals (buckets).

3.13.1.2 DisallowOutliers Method

```
void DisallowOutliers(void)
```

The DisallowOutliers method prevents the recording mechanism from inhibiting the recalling of the array by data values that are outside the current data bounds.

3.13.1.3 RecordDataPoint Method

```
void RecordDataPoint(DWORD dwDP)
```

Name	Type	Use
dwDP	DWORD	The data value to be recorded in the data set.

Use the RecordDataPoint method to add a new data point to the data set being recorded.

3.13.1.4 GetFrequency Method

```
DWORD GetFrequency(int Ordinate, DWORD *lo, DWORD *hi)
```

Name	Type	Use
iOrdinate	int	The index (zero based) of the data interval that is being queried.
lo	DWORD *	Pointer to a DWORD to be filled with the lower bound of the data recorded in this interval.
hi	DWORD *	Pointer to a DWORD to be filled with the upper bound of the data recorded in this interval.

This method will return the frequency count of a particular data interval in the array. The index of the cell can be from 0 to (Number of intervals – 1). The method will also return the lower and upper bounds of the data values that have been recorded in this interval (bucket).

3.13.1.5 GetLowOutlierCount Method

```
int GetLowOutlierCount(void)
```

This method will return the count of low (below the lower data bound) outliers currently recorded in the array.

3.13.1.6 GetHighOutlierCount Method

```
int GetHighOutlierCount(void)
```

This method will return the count of high (above the upper data bound) outliers currently recorded in the array.

3.13.1.7 GetLowOutlier Method

```
DWORD GetLowOutlier(int Ordinate)
```

Name	Type	Use
iOrdinate	int	The index (zero based) of the data outlier that is being queried.

This method will return a DWORD containing the value of a low outlier from the data set. The index may be from 0 to (GetLowOutlierCount – 1).

3.13.1.8 GetHighOutlier Method

```
DWORD GetHighOutlier(int Ordinate)
```

Name	Type	Use
iOrdinate	Int	The index (zero based) of the data outlier that is being queried.

This method will return a DWORD containing the value of a high outlier from the data set. The index may be from 0 to (GetLowOutlierCount – 1).

3.13.2 Usage Notes

The class provides a very simple interface to use for recording data frequencies and recalling the values for display in a tabular or even graphic histogram form.

3.14 NEResolver Class

Header File: DXCommon/NE/NEResolver.h

IMPORTANT NOTE: This class is undergoing further development and may change radically in future releases of DXCommon.

IMPORTANT NOTE: This class uses the Windows LDAP API implementation rather than the Domino LDAP API this precludes the class from being available across platforms. The reason for using the Windows API has nothing to do with the functionality contained in the resolver class itself. It was due to the need to use it an an Authentication DSAPI implementation where the LDAP session needed to be shared between the DSAPI and the resolver and there are some BIND capabilities that were needed that were only exposed by the Windows API. In future releases of the class if a hack can be found to remove the dependency on the Windows LDAP API then the class will revert to using the Domino LDAP API.

The class provides mechanisms for mapping identities “across the great divide” i.e. between a Domino infrastructure and a Microsoft AD infrastructure. The functionality of the class supports a simple usage model, given an arbitrary identity (Domino or AD) the resolver will return a specific identity from the Domino or AD domains that belongs to the same entity as identified by the arbitrary identity that was supplied.

3.14.1 API

3.14.1.1 Constructor

```
NEResolver(ExecEnvironment *xeParent, int iThreadID)
```

Name	Type	Use
xeParent	ExecEnvironment *	Pointer to the current runtime object
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is creating the object.

Before making calls to the NEResolver API the object must be conditioned, the following describe the steps that must be followed to prepare the object for use.

Firstly call the SetDomRefServer method passing it the name of the Domino server that will be used to interrogate the Domino Directory.

Secondly establish a bound LDAP session with an AD LDAP server and pass the LDAP session handle to the object using a call to the setLDAPSessionHandle method.

Last make a call to the setLDAPBaseDN method in order to set the base Distinguished Name that will be used in LDAP searches of the AD directory.

3.14.1.2 setDomRefServer Method

```
BOOL setDomRefServer(char *szServer)
```

Name	Type	Use
szServer	char *	Pointer to a null terminated character string that contains the abbreviated name of the Domino server that will be used to perform directory lookups.

A call to this function will set the Domino server that is to be used for directory lookups. The function will return TRUE if the server is acceptable and FALSE if not. This must be done before attempting to resolve any identities with the resolver.

3.14.1.3 setLDAPSessionHandle Method

```
void setLDAPSessionHandle(LDAP *hlsSet)
```

Name	Type	Use
hlsSet	LDAP *	A valid pointer to an LDAP instance that is already bound to an LDAP AD Server.

Use this method to pass a valid, bound LDAP session handle into the NEResolver. This must be done before attempting to resolve any identities with the resolver.

3.14.1.4 setLDAPBaseDN Method

```
BOOL setLDAPBaseDN(char *szDN)
```

Name	Type	Use
szDN	char *	Pointer to a null terminated character string that contains the distinguished name that is to be used as the search base for queries in the Active Directory.

Use this method to set the Base DN that will be used for searches in AD. It is possible to set wildcards for particular level(s) in the hierarchy using this call. Wildcards are not permitted in the Base DN according to LDAP standards and implementations, the resolver implements code that permits wildcards to be used. Setting the Base DN must be done before attempting to resolve any identities with the resolver.

3.14.1.5 getLDAPSessionHandle Method

```
LDAP *getLDAPSessionHandle(void)
```

This function will return the pointer to the LDAP instance (session handle) that is in use by the NEResolver object.

3.14.1.6 getAffinity Method

```
int getAffinity(char *szAID, int iThreadID)
```

Name	Type	Use
szAID	char *	A pointer to a null terminated character string that contains an arbitrary Domino or AD identity.
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is making the call.

Calls to the getAffinity method pass an arbitrary identity and returns an integer that defines the name space affinity of the identity that was passed. The following symbolic values define the possible values for the affinity.

- `NE_ID_EMPTY` – The identity string is empty, does not contain an identity.
- `NE_ID_UNRESOLVED` - The identity cannot yet be determined for this identity.
- `NE_ID_DOMINO` – The identity passed is a valid Domino identity.
- `NE_ID_AD` – The identity passed is a valid Active Directory identity.
- `NE_ID_EMAIL` – The identity passed is a valid SMTP e-Mail address, this implies that it may be resolvable in both the Domino and the AD name spaces.
- `NE_ID_UNRESOLVEABLE` – It is not possible to resolve an affinity for the passed identity.

3.14.1.7 getDomDN Method

```
int   getDomDN(char *szAID, char *szDomDN, int iDomDNSize, int iThreadID)
```

Name	Type	Use
szAID	char *	A pointer to a null terminated character string that contains an arbitrary Domino or AD identity.
szDomDN	char *	A pointer to a buffer that will receive the Domino Distinguished Name that applies to the entity with the identity passed to the call.
iDomDNSize	int	The size of the buffer that is to receive the Domino DN.
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is making the call.

Given an arbitrary identity passed to the call the method will attempt to resolve the identity either directly or indirectly in the Domino name space and return the Domino Distinguished Name. The call will return one of the following values indicating the result of the call.

- `NERET_INVPARM` – Invalid parameters were passed to the call, nothing was resolved.

- `NERET_RESOLVED` – The identity was fully resolved the Domino DN is valid.
- `NERET_NOTFOUND` – No Domino identity could be found for the supplied identity.
- `NERET_UNRESOLVED` – The passed identity could not be resolved.

3.14.1.8 getDomAN Method

```
int    getDomAN(char *szAID, char *szDomAN, int iDomANSize, int iThreadID)
```

Name	Type	Use
szAID	char *	A pointer to a null terminated character string that contains an arbitrary Domino or AD identity.
szDomAN	char *	A pointer to a buffer that will receive the Domino Abbreviated Name that applies to the entity with the identity passed to the call.
iDomANSize	int	The size of the buffer that is to receive the Domino abbreviated name.
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is making the call.

Given an arbitrary identity passed to the call the method will attempt to resolve the identity either directly or indirectly in the Domino name space and return the Domino Abbreviated Name. The call will return one of the following values indicating the result of the call.

- `NERET_INVPARM` – Invalid parameters were passed to the call, nothing was resolved.
- `NERET_RESOLVED` – The identity was fully resolved the Domino Abbreviated Name is valid.
- `NERET_NOTFOUND` – No Domino identity could be found for the supplied identity.
- `NERET_UNRESOLVED` – The passed identity could not be resolved.

3.14.1.9 getDomCN Method

```
int    getDomCN(char *szAID, char *szDomCN, int iDomCNSize, int iThreadID)
```

Name	Type	Use
szAID	char *	A pointer to a null terminated character string that contains an arbitrary Domino or AD identity.
szDomCN	char *	A pointer to a buffer that will receive the Domino Common Name that applies to the entity with the identity passed to the call.
iDomCNSize	int	The size of the buffer that is to receive the Domino common name.
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number

		of the thread that is making the call.

Given an arbitrary identity passed to the call the method will attempt to resolve the identity either directly or indirectly in the Domino name space and return the Domino Common Name. The call will return one of the following values indicating the result of the call.

- `NERET_INVPARM` – Invalid parameters were passed to the call, nothing was resolved.
- `NERET_RESOLVED` – The identity was fully resolved the Domino Common Name is valid.
- `NERET_NOTFOUND` – No Domino identity could be found for the supplied identity.
- `NERET_UNRESOLVED` – The passed identity could not be resolved.

3.14.1.10 getADDN Method

```
int   getADDN(char *szAID, char *szADDN, int iADDNSize, int iThreadID)
```

Name	Type	Use
szAID	char *	A pointer to a null terminated character string that contains an arbitrary Domino or AD identity.
szADDN	char *	A pointer to a buffer that will receive the AD Distinguished Name that applies to the entity with the identity passed to the call.
iADDNSize	int	The size of the buffer that is to receive the AD DN.
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is making the call.

Given an arbitrary identity passed to the call the method will attempt to resolve the identity either directly or indirectly in the AD name space and return the AD Distinguished Name. The call will return one of the following values indicating the result of the call.

- `NERET_INVPARM` – Invalid parameters were passed to the call, nothing was resolved.
- `NERET_RESOLVED` – The identity was fully resolved the AD DN is valid.
- `NERET_NOTFOUND` – No AD identity could be found for the supplied identity.
- `NERET_UNRESOLVED` – The passed identity could not be resolved.

3.14.1.11 getADCN Method

```
int   getADCN(char *szAID, char *szADCN, int iADCNSize, int iThreadID)
```

Name	Type	Use
szAID	char *	A pointer to a null terminated character string that contains an arbitrary Domino or AD identity.
szADCN	char *	A pointer to a buffer that will receive the AD Common Name that applies to the entity with the identity passed to the call.
iADCNSize	int	The size of the buffer that is to receive the AD CN.

iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is making the call.

Given an arbitrary identity passed to the call the method will attempt to resolve the identity either directly or indirectly in the AD name space and return the AD Common Name. The call will return one of the following values indicating the result of the call.

- `NERET_INVPARM` – Invalid parameters were passed to the call, nothing was resolved.
- `NERET_RESOLVED` – The identity was fully resolved the AD CN is valid.
- `NERET_NOTFOUND` – No AD identity could be found for the supplied identity.
- `NERET_UNRESOLVED` – The passed identity could not be resolved.

3.14.1.12 getADUID Method

```
int    getADUID(char *szAID, char *szADUID, int iADUIDSize, int iThreadID)
```

Name	Type	Use
szAID	char *	A pointer to a null terminated character string that contains an arbitrary Domino or AD identity.
szADUID	char *	A pointer to a buffer that will receive the AD UID that applies to the entity with the identity passed to the call.
iADUIDSize	int	The size of the buffer that is to receive the AD UID.
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is making the call.

Given an arbitrary identity passed to the call the method will attempt to resolve the identity either directly or indirectly in the AD name space and return the AD UID. The call will return one of the following values indicating the result of the call.

- `NERET_INVPARM` – Invalid parameters were passed to the call, nothing was resolved.
- `NERET_RESOLVED` – The identity was fully resolved the AD UID is valid.
- `NERET_NOTFOUND` – No AD identity could be found for the supplied identity.
- `NERET_UNRESOLVED` – The passed identity could not be resolved.

3.14.1.13 getADLogin Method

```
int    getADLogin(char *szAID, char *szADCN, int iADCNSize, int iThreadID)
```

Name	Type	Use
szAID	char *	A pointer to a null terminated character string that contains an arbitrary Domino or AD identity.
szADLogin	char *	A pointer to a buffer that will receive the AD Login Name that applies to the entity with the identity passed to the call.

iADLoginSize	int	The size of the buffer that is to receive the AD Login Name.
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is making the call.

Given an arbitrary identity passed to the call the method will attempt to resolve the identity either directly or indirectly in the AD name space and return the AD Login Name. The call will return one of the following values indicating the result of the call.

- `NERET_INVPARM` – Invalid parameters were passed to the call, nothing was resolved.
- `NERET_RESOLVED` – The identity was fully resolved the AD Login Name is valid.
- `NERET_NOTFOUND` – No AD identity could be found for the supplied identity.
- `NERET_UNRESOLVED` – The passed identity could not be resolved.

3.14.1.14 getEmailAddress Method

```
int getEmailAddress(char *szAID, char *szEMAIL, int iEMSize, int iThreadID)
```

Name	Type	Use
szAID	char *	A pointer to a null terminated character string that contains an arbitrary Domino or AD identity.
szEMAIL	char *	A pointer to a buffer that will receive the Email address that applies to the entity with the identity passed to the call.
iEMAILSize	int	The size of the buffer that is to receive the Email address.
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is making the call.

Given an arbitrary identity passed to the call the method will attempt to resolve the identity either directly or indirectly in the AD and or Domino name spaces and return the SMTP Email address. The call will return one of the following values indicating the result of the call.

- `NERET_INVPARM` – Invalid parameters were passed to the call, nothing was resolved.
- `NERET_RESOLVED` – The identity was fully resolved the Email address is valid.
- `NERET_NOTFOUND` – No AD or Domino identity could be found for the supplied identity.
- `NERET_UNRESOLVED` – The passed identity could not be resolved.

3.14.1.15 ShowDuplicates Member

The BOOL ShowDuplicates member of the class can be set to TRUE which will cause the NEResolver to show messages if duplicate keys are encountered during the resolution process. The resolver will NOT resolve any entity that matches duplicate keys anywhere on the resolution path.

3.14.2 Usage Notes

Refer to the Person Look Up (PLU) application for an example of deploying the NEResolver in a live application setting.

4. Supporting Classes

This section describes the different states that a transaction moves through during normal operation and fault handling.

4.1 CopyRequest Class

Header File: DXCommon/DBC/CopyRequest.h

The CopyRequest class is used to pass information to a DbCopier object, the information controls theee copying of a single Notes database.

4.1.1 API

4.1.1.1 Constructor

```
CopyRequest(void)
```

4.1.1.2 PermitRun Member

```
BOOL        PermitRun
```

The RunPermit must be set to TRUE, if it is set to FALSE at any point during the execution of a copy operation then the DbCopier will immediately terminate the copy operation. This mechanism can be used in asynchronous copy operations to abort a copy operation.

4.1.1.3 bAllowProxyFolderMessages Member

```
BOOL        bAllowProxyFolderMessages
```

If set to TRUE then the DbCopier will pass all messages that are generated during Folder maintenance operations through the ProxyLogMessage interfacte. The ProxyLogMessage interface allows applications to intercept log messages that are generated by the DbCopier.

4.1.1.4 ReturnCode Member

```
int                 ReturnCode
```

The ReturnCode member will contain a value indicating the completion status of the requested copy operation. The status can be any one of the following values.

- `RETURN_NOERROR` – The request was completed without any errors.

- `RETURN_CWARN` – The request was completed but one or more warning messages were issued.
- `RETURN_CERR` - The request was completed but one or more errors were encountered.
- `RETURN_INCOMPLETE` – The request could not be completed.

4.1.1.5 AllowRetry Member

```
BOOL        AllowRetry
```

The AllowRetry member is used to indicate if (TRUE) a request can be retried if it fails. The retry mechanism must be implemented by the application that invokes the DbCopier, the application should maintain the RetryCount member incrementing it from zero for each time that a request is retried, this mechanism is implemented by default if using the DX TransactionHandler. The DbCopier will recognise a request that is being re-tried and will execute any pre-processing steps that are indicated in the RestartActions member of the request.

4.1.1.6 RestartActions Member

```
int                  RestartActions
```

If a request is being retried (indicated by a non-zero value in the RetryCount) then this member will indicate one or more actions that should be undertaken prior to performing the copy operation. The field can contain any of the following values that may be combined using OR.

- `RESTART_DEL_TARGET` – Delete the target database if it exists.
- `RESTART_ASSERT_FIXUP` – Force a fixup of the source database before copying.

4.1.1.7 PreProcessActions Member

```
int                  PreProcessActions
```

This member may be set to a value of `PRESTART_DEL_TARGET` to cause the DbCopier to delete the target database before performing the copy operation.

4.1.1.8 FaultTolerance Member

```
int                  FaultTolerance
```

The FaultTolerance member may be set to one or more of the following values, the values indicate particular anomalous processing conditions that will be treated as warnings rather than errors that would cause the request to be terminated. Multiple values can be combined using OR.

- `ALLOW_FOLDER_ERRORS` – Errors during folder maintenance will be treated as warnings.

- `ALLOW_ACCESS_ERRORS` – If the DbCopier does not have read access to individual documents then the copy operation will proceed and drop these documents with a warning.
- `ALLOW_COPY_ERRORS` – If an error is encountered while copying an individual document the copy operation will continue, use this setting to drop corrupt documents.
- `RECOVER_RSRC_ERRORS` – this setting allows the DbCopier to attempt an automated recovery from problems that result from a shortage of resources.

4.1.1.9 CopyReplica Member

```
BOOL        CopyReplica
```

Set the CopyReplica to TRUE to cause the target database to be a replica copy of the source database. Set it to FALSE to create a non-replica copy.

4.1.1.10 CopyStubs Member

```
BOOL        CopyStubs
```

Set the CopyStubs member to TRUE to cause deletion stubs encountered in the source database to be copied to the target database, this setting is normally only useful when creating a replica copy. Set the value to FALSE to have the DbCopier skip over any deletion stubs in the source database.

4.1.1.11 CopyPrivates Member

```
BOOL        CopyPrivates
```

Set the CopyPrivates member to TRUE to have the DbCopier attempt to copy any private design elements encountered in the source database to the target database. Set the value to FALSE to have the DbCopier skip over any private design elements encountered in the source.

4.1.1.12 RefreshDesign Member

```
BOOL        RefreshDesign
```

Set this member to TRUE to have the DbCopier force a design refresh of the target database after the copying operations are completed. It is possible to have the copier set a new template name that will be used for the design refresh. The corresponding template database must be in place on the target server for this operation to succeed. Set the value to FALSE if no refresh is required.

4.1.1.13 CopyFTI Member

```
BOOL        CopyFTI
```

Set this member to TRUE to have the DbCopier copy any full text index settings on the source database to the target database, the Full Text Index will not be built on the target database this will be done by the normal Index Maintenance processes on the target server. Set the value to false to ignore any full text index settings.

4.1.1.14 CopyAndBuildFTI Member

```
BOOL        CopyAndBuildFTI
```

Set this member to TRUE to have the DbCopier copy any full text index settings on the source database to the target database and immediately build the Full Text Index, otherwise set the value to FALSE.

4.1.1.15 BuildIndexes Member

```
BOOL        BuildIndexes
```

Set this member to TRUE to cause the DbCopier to build all of the view indexes in the target database after performing the copy operations, otherwise set the value to FALSE.

4.1.1.16 EnableTXLogging Member

```
BOOL        EnableTXLogging
```

Set this member to TRUE to have the DbCopier enable Transaction Logging on the target database.

Set the value to FALSE to have Transaction Logging remain disabled after the copy completes.

4.1.1.17 ConsistentACLOff Member

```
BOOL        ConsistentACLOff
```

Set this member to TRUE to have the DbCopier turn off the Enforce Consistent ACL property in the target database. Set the value to FALSE to leave the setting as it was copied from the source database.

4.1.1.18 CopyFolders Member

```
BOOL        CopyFolders
```

This setting only applies to non-replica copies, if set to TRUE it will cause the DbCopier to populate the folders of the target database with the same content as the corresponding folders in the source database.

Set the value to FALSE to leave the folders in the target database unpopulated. Folder content is always populated during a replica copy operation.

4.1.1.19 ForceFixup Member

```
BOOL          ForceFixup
```

Setting this member to TRUE will cause the DbCopier to perform a fixup on the source database before performing the copy operations. Set the value to FALSE if no fixup is required.

4.1.1.20 DisableAgents Member

```
BOOL          DisableAgents
```

Set this member to TRUE to prevent scheduled agents from being executed in the target database. The setting does not disable individual agents instead it turns off the database level property that is used by the agent manager to determine if it should schedule agents in the database, when set off the Agent Manager effectively ignores the database. Set the value to FALSE to allow any scheduled agents in the target database to be executed by the Agent Manager, providing that the agent signer can execute agents on the target server.

4.1.1.21 OneTimeReplicate Member

```
BOOL          OneTimeReplicate
```

This setting only applies to replica copy operations. If the value is set to TRUE then once the copy operations have completed the DbCopier will perform a one-time replication between the source and target database. Set the value to FALSE if the replication is not required.

4.1.1.22 szTransID Member

```
char          szTransID[MAX_DATABASE + 1]
```

This member can be set to an arbitrary null terminated character string. The DbCopier will only use this value for reporting purposes.

4.1.1.23 szSourceServer Member

```
char          szSourceServer[MAX_SERVER + 1]
```

Set this member to a null terminated character string containing the abbreviated name of the server where the source database can be found. Set the string to “Local” or the empty string “” if the source database is on the same server where the application is running.

4.1.1.24 szSourceDatabase Member

```
char        szSourceDatabase[MAX_DATABASE + 1]
```

Set this member to a null terminated character string containing the name of the source database (relative to the Notes Data Directory).

4.1.1.25 szSourceDBReplicaID Member

```
char        szSourceDBReplicaID[MAX_DESC + 1]
```

This member can optionally be set to a null terminated character string containing a character representation of the Replica ID of the source database. If a value is present then the DbCopier will perform a check that the source database has a Replica ID that matches the value provided, if the values do not match then the transaction is failed.

4.1.1.26 szTargetServer Member

```
char        szTargetServer[MAX_SERVER + 1]
```

Set this member to a null terminated character string containing the abbreviated name of the server where the target database will be placed. Set the string to “Local” or the empty string “” if the target database will be on the same server where the application is running.

4.1.1.27 szTargetDatabase Member

```
char        szTargetDatabase[MAX_DATABASE + 1]
```

Set this member to a null terminated character string containing the name of the target database (relative to the Notes Data Directory).

4.1.1.28 szGroupToAdd Member

```
char        szGroupToAdd[MAX_GROUP + 1]
```

This member can optionally be set to a null terminated character string containing the name of a group that will be added to the ACL of the target database. The group will be added as a mixed group with Manager access, all rights enabled and all Roles that exist in the ACL will be asserted.

4.1.1.29 szTemplateName Member

```
char          szTemplateName[MAX_TEMPLATE + 1]
```

This member can optionally be set to a null terminated character string containing a Design Template name from which the target database will inherit design. A design refresh can be enforced with the new template by setting the RefreshDesign member to TRUE, otherwise the design will be updated by the Design Task if it is running on the target server.

4.1.1.30 szAgentName Member

```
char          szAgentName[MAX_ELEMENT + 1]
```

This member can optionally be set to a null terminated character string containing the name of an agent in the target database, if a value is present then the DbCopier will run the agent after all copy operations have completed. The agent can be used to “condition” documents in the target database after it has been copied performing such operations as changing name lists in readers and authors fields.

4.1.1.31 Processing Statistics Members

The following members are exposed to give applications access to statistical information about copy performance.

- `DWORD dwDataDocsCopied` – Count of data documents copied.
- `DWORD dwDataStubsCopied` – Count of data deletion stubs copied.
- `DWORD dwDataAccessErrors` – Count of documents in the source database that the copier did not have reader access to.
- `DWORD dwDataOtherErrors` – Count of notes that reported an error while being copied.
- `DWORD dwDataRetries` – Count of note copy operations that were retried.
- `DWORD dwDesignDocsCopied` – Count of design documents copied.
- `DWORD dwDesignStubsCopied` – Count of design deletion stubs copied.
- `DWORD dwDesignAccessErrors` – Count of design documents in the source database that the copier did not have reader access to.
- `DWORD dwDesignOtherErrors` – Count of design notes that reported an error while being copied.
- `DWORD dwDesignRetries` – Count of design note copy operations that were retried.
- `DWORD dwCopyPhaseElapsed` – Elapsed time of the copy phase of processing in seconds.
- `DWORD dwFoldersCopied` – The number of folders maintained.
- `DWORD dwFolderDocs` – The total number of documents that were added to all folders.
- `DWORD dwFolderPhaseElapsed` – The elapsed time for folder (and FTI) processing in seconds.

- `DWORD dwDesignPhaseElapsed` – The elapsed time spent doing design refresh.
- `DWORD dwAgentPhaseElapsed` – The elapsed time spent running the agent.
- `DWORD dwIndexPhaseElapsed` – The elapsed time spent building view indexes.
- `DWORD dwReplPhaseElapsed` – The elapsed time spent replicating the database.
- `DWORD dwTotalElapsed` – The total elapsed time spent on copy processing in seconds.
- `DWORD dwSizeKB` – The final size of the target database in Kb.

4.1.2 Usage Notes

To invoke a copy operation asynchronously on a DbCopier object follow the steps below.

1) Create a CopyRequest object and populate it with the parameters needed for the copy operation.
2) Create a PartCopyRequest object and set the address of the CopyRequest object in the “Parent” member and set the “CopyAction” member to “COPY_DATABASE”.
3) Use the “PostARequest” method on the runtime object to dispatch the “PartCopyRequest” message to the DbCopier.

4.2 PartCopyRequest Class

Header File: DXCommon/DBC/PartCopyRequest.h

The PartCopyRequest class is mainly used by the DbCopier call to initiate and control the asynchronous copy micro-operation that are used during a database copy operation. Applications would only use the class to create an object to asynchronously initiate a copy operation.

4.2.1 API

4.2.1.1 Constructor

`PartCopyRequest(CopyRequest *crOriginator)`

Name	Type	Use
crOriginator	CopyRequest *	Pointer to the CopyRequest object that is the parent of this PartCopyRequest.

4.2.1.2 Parent Member

```
CopyRequest *Parent
```

This member holds the address of the parent CopyRequest object that is supplied to the constructor.

4.2.1.3 pcrqCopyDesign Member

```
PartCopyRequest *pcrqCopyDesign
```

If this PartCopyRequest is a request to copy design notes from the source to the target database then this member contains the address of the intermediate PartCopyRequest for copying the design.

4.2.1.4 pcrqCopyData Member

```
PartCopyRequest *pcrqCopyData
```

If this PartCopyRequest is a request to copy data notes from the source to the target database then this member contains the address of the intermediate PartCopyRequest for copying the data.

4.2.1.5 dbcCopier Member

```
void *dbcCopier
```

This member contains the address of the DbCopier object that is performing the copy.

4.2.1.6 CopyAction Member

```
int CopyAction
```

This member contains the action that is to be carried out by the DbCopier when it receives this request, actions may be any of the following symbolic values.

- `COPY_DATABASE` – This request carries a top level copy request that should now be executed.
- `COPY_DESIGN` - The request is to copy the complete database design.
- `COPY_DESIGN_NOTES` – The request is to copy a number of individual design notes.
- `COPY_DATA` – The request is to copy all of the data documents in the database.
- `COPY_DATA_NOTES` – The request is to copy a number of individual data notes.
- `COPY_MAINTAIN_FOLDERS` – The request is to maintain all of the folders in the target database.

- `COPY_MAINTAIN_FOLDER` - The request is to maintain the contents of a single folder.
- `COPY_FTI` – The request is to copy the Full Text Index definition and optionally build the index.
- `UPGRADE_DESIGN` – The request is to update the design of the target database.
- `COPY_BUILD_INDEXES` – The request is to build all of the view indexes in the target database.
- `COPY_BUILD_INDEX` – The request is to build an individual view index in the target database.
- `RUN_AGENT` - The request is to run an agent in the target database.

4.2.1.7 CompletionStatus Member

```
int CompletionStatus
```

This member contains the completion status of the request, after processing. The status can be any of the following symbolic values.

- `ACTION_COMPLETED` – The request was completed successfully.
- `ACTION_INPROGRESS` – The request is currently being executed.
- `ACTION_WARN` – The request was completed but one or more warnings were issued.
- `ACTION_FAILED` – The request has failed.
- `ACTION_RECOVER` – The request has failed but recovery should be possible.
- `ACTION_ABORT` – The request has failed all other requests should be quenched or discarded and the top level copy request should be marked in error.

4.2.1.8 iProcThread Member

```
int   iProcThread
```

This member contains the identity of the thread that is or was processing the request.

4.2.1.9 nidToCopy Array

```
NOTEID nidToCopy[COPY_NOTE_MULTIPLEX]
```

For operations that copy notes this member contains the array of Note IDs that are to be copied, the count of notes in the array is contained in the CopyNoteCount member.

4.2.1.10 CopyNoteCount Member

```
int CopyNoteCount
```

This member contains the count of Note IDs in the nidToCopy array.

4.3 MoveRequest Class

Header File: DXCommon/DBC/MoveRequest.h

The MoveRequest class is used to pass information that controls the operation of a Database Mover (DbMover) engine.

The DbMover engine was originally developed to solve the problem of redistributing notes databases across multiple storage volumes without shutting down the Domino server or limiting the use of databases while they are being moved between storage volumes. Having addressed the problem of storage redistribution it was noted that DbMover had other capabilities such as creating media backups and deleting databases already plumbed in and so minor code changes turned it into a more generic Notes database storage management tool.

The DbMover engine recognises three primitive actions that can be combined in different ways to accomplish different tasks. The primitive actions are Copy (C) which makes an online copy of an open (transaction logged) database, Delete (D) which deletes a database while it is online and Link (L) which creates a database link in place of the original database pointing to a new storage location (which can be outside of scope of the Notes Data Directory).

4.3.1 API

4.3.1.1 Constructor

```
MoveRequest(void)
```

4.3.1.2 PermitRun Member

```
BOOL        PermitRun
```

The RunPermit must be set to TRUE, if it is set to FALSE at any point during the execution of a move operation then the DbMover will immediately terminate the move operation. This mechanism can be used in asynchronous move operations to abort the operation.

4.3.1.3 ReturnCode Member

```
int                 ReturnCode
```

The ReturnCode member will contain a value indicating the completion status of the requested move operation. The status can be any one of the following values.

- `RETURN_NOERROR` – The request was completed without any errors.

- `RETURN_CWARN` – The request was completed but one or more warning messages were issued.
- `RETURN_CERR` - The request was completed but one or more errors were encountered.
- `RETURN_INCOMPLETE` – The request could not be completed.

4.3.1.4 AllowRetry Member

```
BOOL        AllowRetry
```

The AllowRetry member is used to indicate if (TRUE) a request can be retried if it fails. The retry mechanism must be implemented by the application that invokes the DbMover, the application should maintain the RetryCount member incrementing it from zero for each time that a request is retried, this mechanism is implemented by default if using the DX TransactionHandler.

4.3.1.5 CopyBfrKb Member

```
UINT        CopyBfrKb
```

If the request is for an operation that includes a copy action then this member will be used to determine the size of the read/write buffer that will be used for the copy process I/O.

4.3.1.6 PerformCopy Member

```
BOOL        PerformCopy
```

Set this member to TRUE if this request includes a copy operation, otherwise set to FALSE.

4.3.1.7 PerformDelete Member

```
BOOL        PerformDelete
```

Set this member to TRUE if this request includes a delete operation otherwise set to FALSE.

4.3.1.8 PerformLink Member

```
BOOL        PerformLink
```

Set this member to TRUE if this request includes a link operation otherwise set to FALSE.

4.3.1.9 szTransID Member

```
char        szTransID[MAX_DATABASE + 1]
```

This member can be set to an arbitrary null terminated character string. The DbMover will only use this value for reporting purposes.

4.3.1.10 szSourceDatabase Member

```
char        szSourceDatabase[MAX_DATABASE + 1]
```

Set this member to a null terminated character string containing the name of the source database (relative to the Notes Data Directory).

4.3.1.11 szTargetDir Member

```
char        szTargetDir[MAX_DATABASE + 1]
```

This member is only applicable to requests that contain a Copy or Link action. Set the member to a null terminated string containing the name of the directory to which the copy will be made.

4.3.1.12 Processing Statistics Members

The following members are exposed to applications to provide some feedback on the performance of DbMover operations.

- `DWORD dwDataDocsCopied` – Count of data documents in the database being processed.
- `DWORD dwTotalElapsed` – The elapsed time of the DbMover operation in seconds.
- `DWORD dwSizeKB` – The size (in Kb) or the database being processed.

4.3.2 Usage Notes

To invoke a Mover operation asynchronously on a DbMover object follow the steps below.

1) Create a MoveRequest object and populate it with the parameters needed for the operation.
2) Create a PartMoveRequest object and set the address of the MoveRequest object in the "Parent" member and set the "MoveAction" member to "MOVE_DATABASE".

Use the "PostARequest" method on the runtime object to dispatch the "PartMoveRequest" message to the DbMover.

4.4 PartMoveRequest Class

Header File: DXCommon/DBC/PartMoveRequest.h

The PartMoveRequest class is mainly used by the DbMover call to initiate and control the asynchronous micro-operation that are used during a database move operation. Applications would only use the class to create an object to asynchronously initiate a move operation.

4.4.1 API

4.4.1.1 Constructor

```
PartMoveRequest(MoveRequest *mrOriginator)
```

Name	Type	Use
mrOriginator	MoveRequest *	Pointer to theMoveRequest object that is the parent of this PartMoveRequest.

4.4.1.2 Parent Member

```
MoveRequest *Parent
```

This member holds the address of the parent MoveRequest object that is supplied to the constructor.

4.4.1.3 dbmMover Member

```
void *dbmMover
```

This member contains the address of the DbMover object that is performing the move.

4.4.1.4 MoveAction Member

```
int MoveAction
```

This member contains the action that is to be carried out by the DbMover when it receives this request, actions may be any of the following symbolic values.

- `MOVE_DATABASE` – This request carries a top level move request that should now be executed.

4.4.1.5 CompletionStatus Member

```
int CompletionStatus
```

This member contains the completion status of the request, after processing. The status can be any of the following symbolic values.

- `ACTION_COMPLETED` – The request was completed successfully.
- `ACTION_INPROGRESS` – The request is currently being executed.
- `ACTION_WARN` – The request was completed but one or more warnings were issued.
- `ACTION_FAILED` – The request has failed.
- `ACTION_ABORT` – The request has failed all other requests should be quenched or discarded and the top level copy request should be marked in error.

4.4.2 Usage Notes

To invoke a Mover operation asynchronously on a DbMover object follow the steps below.

1) Create a MoveRequest object and populate it with the parameters needed for the operation.
2) Create a PartMoveRequest object and set the address of the MoveRequest object in the “Parent” member and set the “MoveAction” member to “MOVE_DATABASE”.

Use the “PostARequest” method on the runtime object to dispatch the “PartMoveRequest” message to the DbMover.

4.5 ModRequest Class

Header File: DXCommon/DBP/ModRequest.h

The ModRequest class is used to pass requests for Database Property changes to a DbModifier engine.

4.5.1 API

4.5.1.1 Constructor

```
ModRequest(void)
```

4.5.1.2 PermitRun Member

```
BOOL        PermitRun
```

The RunPermit must be set to TRUE, if it is set to FALSE at any point during the execution of a modify operation then the DbModifier will immediately terminate the modify operation. This mechanism can be used in asynchronous modify operations to abort a modify operation.

4.5.1.3 hDbSrc Member

```
DBHANDLE    hDBSrc
```

The hDbSrc member contains the database handle of the database that is to be modified.

4.5.1.4 ReturnCode Member

```
int                  ReturnCode
```

The ReturnCode member will contain a value indicating the completion status of the requested modify operation. The status can be any one of the following values.

- `RETURN_NOERROR` – The request was completed without any errors.
- `RETURN_CWARN` – The request was completed but one or more warning messages were issued.
- `RETURN_CERR` - The request was completed but one or more errors were encountered.
- `RETURN_INCOMPLETE` – The request could not be completed.

4.5.1.5 AllowRetry Member

```
BOOL        AllowRetry
```

The AllowRetry member is used to indicate if (TRUE) a request can be retried if it fails. The retry mechanism must be implemented by the application that invokes the DbModifier, the application should maintain the RetryCount member incrementing it from zero for each time that a request is retried, this mechanism is implemented by default if using the DX TransactionHandler.

4.5.1.6 dwPropsToMod Member

```
DWORD       dwPropsToMod
```

The dwPropsToMod is set to a mask of bits indicating which properties are to be modified by this request, the mask is constructed by ORing together any combination of the following values.

- `MOD_PROP_FTI` – Modify the Full Text Index settings.
- `MOD_PROP_BGA` – Modify the Background Agents property.

- `MOD_PROP_CAL` – Modify the Calendar property.
- `MOD_PROP_REPT` – Modify the Disable Replication (Temporarily) property.
- `MOD_PROP_REPP` – Modify the Disable Replication (Permanently) property.
- `MOD_PROP_BROWSE` – Modify the Database can be Browsed property.
- `MOD_PROP_CAT` – Modify the the show database in catalog property.
- `MOD_PROP_TXL` – Modify the Database is Transaction Logged property.
- `MOD_PROP_CACL` – Modify the Enforce Consistent ACL property.
- `MOD_PROP_MIA` – Modify the Maximum Internet Access Level property.
- `MOD_PROP_TPLT` – Modify the Template name that the database inherits design from.
- `MOD_PROP_QUOTA` – Modify the Database Quota settings.

4.5.1.7 dwPropSettings Member

`DWORD          dwPropSetting`

The dwPropSettings member contains the ON or OFF value for the property settings that are binary, any of the following values can be combined using OR. If a particular bit in the member is ON then the corresponding property is set on.

- `MOD_PROP_FTI` – If ON the database has a Full Text Index if OFF then it does not. This modifies the DBOPTION_FT_INDEX database property.
- `MOD_PROP_BGA` – If ON then background agents will be executed in the database if OFF then they will not. This modifies the REPLFLG_NO_CHRONOS setting in the replication flags.
- `MOD_PROP_CAL` – If ON then the database will be marked as containing a claendar and hence will be processed by the schedule manager, if OFF then it will not. This modifies the DBOPTION_HAS_CALENDAR database property.
- `MOD_PROP_REPT` – If ON the replication will be disabled in the database, this setting is temporary and can be set back on from the client. If off the replication is enabled. This modifies the REPLFLG_DISABLE setting in the replication flags.
- `MOD_PROP_REPP` – If ON then replication will be disabled in the database, this setting cannot be modified from the client. If OFF then replication will be enabled. This modifies the REPLFLG_NEVER_REPLICATE setting in the replication flags.
- `MOD_PROP_BROWSE` – If ON then the database will not be browsable. If OFF then the database will be browsable. This modifies the REPLFLG_DO_NOT_BROWSE setting in the replication flags.
- `MOD_PROP_CAT -` If ON then the database will not be listed in the database catalog, if OFF then the database will be listed in the catalog. This modifies the REPLFLG_DO_NOT_CATALOG setting in the replication flags.
- `MOD_PROP_TXL` – If ON the the database will be Transaction Logged, if OFF then the database will not be transaction logged.
- `MOD_PROP_CACL` – If On then consistent ACL will be enforced across all replicas of the database, if OFF it will not be enforced. This modifies the DBOPTION_UNIFORM_ACCESS database property.

4.5.1.8 MaxInternetAccess Member

```
WORD        MaxInternetAccess
```

This member determines the maximum internet access level set for the database. Set this to one of the valid ACL_LEVEL_xxx symbolic values (see Notes API documentation).

4.5.1.9 QuotaWarn Member

```
DWORD       QuotaWarn
```

The QuotaWarn member is used to set the database warning threshold level in the database. The value is specified in Kb units.

4.5.1.10 QuotaLimit Member

```
DWORD       QuotaLimit
```

The QuotaLimit Member is used to set the database quota limit for the database. The value is specified in Kb units.

4.5.1.11 szTransID Member

```
char        szTransID[MAX_DATABASE + 1]
```

This member can be set to an arbitrary null terminated character string. The DbModifier will only use this value for reporting purposes.

4.5.1.12 szSourceServer Member

```
char        szSourceServer[MAX_SERVER + 1]
```

Set this member to a null terminated character string containing the abbreviated name of the server where the source database can be found. Set the string to “Local” or the empty string “” if the source database is on the same server where the application is running.

4.5.1.13 szSourceDatabase Member

```
char        szSourceDatabase[MAX_DATABASE + 1]
```

Set this member to a null terminated character string containing the name of the source database (relative to the Notes Data Directory).

4.5.1.14 szTemplateName Member

```
char        szTemplateName[MAX_TEMPLATE + 1]
```

This member can optionally be set to a null terminated character string containing a Design Template name from which the target database will inherit design. The design will be updated by the Design Task if it is running on the source server.

4.6 DXACLRule Class

Header File: DXCommon/ACL/DXACLRule.h

The DXACLRule class is used to contain information about a single rule that is used to determine the conformance of individual ACL entries.

4.6.1 API

4.6.1.1 Constructor

```
DXACLRule(ExecEnvironment *xeParent, int iThreadID)
```

Name	Type	Use
xeParent	ExecEnvironment *	Pointer to the current runtime object
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is creating the object.

4.6.1.2 matchEntry Method

```
BOOL matchEntry(char *szEntry, WORD wFlags, int iThreadID)
```

Name	Type	Use
szEntry	char*	Pointer to a null terminated character string containing the

		name from an ACL entry.
wFlags	WORD	The ACL flags that apply to the entry being matched.
iThreadID	int	For single threaded applications always specify 0 to indicate the main thread of a program otherwise specify the number of the thread that is making the call.

The matchEntry method takes information from an ACL entry and determines if current ACL rule matches the information passed. The method returns TRUE if the entry matches and FALSE if it does not.

4.6.1.3 checkSettings Method

```
int     checkSettings(WORD wAccessLevel, ACL_PRIVILEGES *apEntry, WORD wAccessFlags,
        int iThreadID)
```

The checkSettings method will take the settings from an ACL entry and determine if they match the settings specified by the current Rule.

The method will return zero if the settings fully conform to the Rule. If the entry does not conform then the return value will be a composite of any of the following bit settings.

- `PS_ACTION_MOD_LEVEL` – The ACL Access level needs to be modified to conform.
- `PS_ACTION_MOD_PRIVS` – The ACL Roles need to be modified to conform.
- `PS_ACTION_MOD_FLAGS` – The ACL flag settings need to be modified to conform.

4.7 DXNoteScanReq Class

Header File: DXCommon/DX/DXNoteScanReq.h

Header File: DXCommon/MTDX/DXNoteScanReq.h

The DXNoteScanReq class is used to create objects that pass scanning requests to a DXNoteScanner database scanning engine.

4.7.1 API

4.7.1.1 Constructor

```
DXNoteScanReq(DXNoteScanTable *nstToUse, int iThreadID)
```

Name	Type	Use
nstToUse	DXNoteScanTable*	Pointer to a populated DXNoteScanTable.
iThreadID	int	For single threaded applications always specify 0 to indicate the

		main thread of a program otherwise specify the number of the thread that is making the call.

The DXNoteScan table is a structure that contains the search requirements and the output members for up to 10 simultaneous searches. The table is made up as follows.

```
typedef struct {
	DXNoteScanEntry		dxnsEntry[MAX_SEARCH_ENTRIES];		//  Array of
DXNoteScanEntry objects
} DXNoteScanTable;
```

Each entry has the following content.

```
typedef struct {
	DHANDLE hitMatches;
	//  Handle of IDTable containing matches for this search
	WORD   wNoteClass;
	//  Note classes to scan for
	DWORD  dwSearchFlags;
	//  Search Flags... see FLAG_NOTE_SCAN_xxx
	char   szSearchString[MAX_SEARCH_STRING];			//  Search
string to be used for this search
	DWORD  dwCount;
	//  Number of noteids added to IDTable
} DXNoteScanEntry;
```

The values should be set as follows.

4.7.1.1.1 DHANDLE hitMatches Member

This member can optionally be set to the handle of an ID Table constructed by the caller, if set to NULLHANDLE then the scanner will construct the ID Table for the caller.

4.7.1.1.2 WORD wNoteClass Member

This member should be set to the note classes that are to be retrieved. See the Notes API NOTE_CLASS_XXX definition for possible values.

4.7.1.1.3 DWORD dwSearchFlags Member

This member can be set to indicate special characteristics of notes that should be searched for during this scan. Any of the following values can be combined using OR to set this value. By default the value should be set to zero.

- FLAG_NOTE_SCAN_OBJECT_FILE – Select notes that contain an attached file object.
- FLAG_NOTE_SCAN_OBJECT_OTHER – Select notes that have attachments other than files.
- FLAG_NOTE_SCAN_SIGNED – Select notes that are signed.

- `FLAG_NOTE_SCAN_ENCRYPTED` – Select notes that are encrypted.
- `FLAG_NOTE_SCAN_DELSTUBS` – Select notes that are deletion stubs.
- `FLAG_NOTE_SCAN_REPCONS` – Select notes that are replication conflicts.
- `FLAG_NOTE_SCAN_NOACCESS` – Select notes to which the scanner does not have reader access.
- `FLAG_NOTE_SCAN_NOT_DELSTUB` – **Exclude** notes that are deletion stubs.

4.7.1.1.4 Char szSearchString Member

This member can optionally specify a null terminated string containing a search string that specifies the notes that are to be selected. The search string is combined with the Note Classes and any search flags specified to determine the notes that will be selected.

4.7.1.1.5 DWORD dwCount Member

This member will be populated by the scanner with the number of Notes that matched the search criteria and therefore have a NoteID in the ID Table.

4.7.1.2 hdbToScan Member

```
DBHANDLE          hdbToScan
```

Set this member to the DBHANDLE of the database that is to be scanned. If this is not set then set the server and database name members and the scanner will opan and close the database for the caller.

4.7.1.3 szServer Member

```
char              szServer[MAXUSERNAME]
```

If the caller is not passing a DBHANDLE then this member should be set to a null terminated character string containing the abbreviated name of the server that stores the database to be scanned. An empty string of the value ”Local” can be used to indicate that the database resides on the server/workstation where the scanner is executing.

4.7.1.4 szDbPath Member

```
char              szDbPath[MAXPATH]
```

If the caller is not passing a DBHANDLE then this member should be set to a null terminated string containing the path of the database that is to be scanned (relative to the Notes Data Directory).

4.7.1.5 dwMaxErrors Member

```
DWORD                 dwMaxErrors
```

Set this value to the maximum number of errors that may be encountered during the scan before the scan is terminated.

4.7.1.6 ReturnCode Member

```
int                   ReturnCode
```

The ReturnCode member will contain a value indicating the completion status of the requested modify operation. The status can be any one of the following values.

- `RETURN_NOERROR` – The request was completed without any errors.
- `RETURN_CWARN` – The request was completed but one or more warning messages were issued.
- `RETURN_CERR` - The request was completed but one or more errors were encountered.
- `RETURN_INCOMPLETE` – The request could not be completed.

4.7.1.7 Processing Statistics Members

The following members are exposed to provide insight into the processing generated by the scan that was processed.

- `dwNotesToScan` – The number of notes returned by the base search.
- `dwNotesScanned` – The total number of notes scanned during processing.
- `dwErrors` – The number of errors encountered during the scanning process.
- `dwElapsed` – The elapsed time in seconds that the scan processing took.

5. Appendixes

5.1 Appendix A: ACL Rule Set XML

The XML document supplied is a well-formed XML document that specifies, in a declarative style, what rules the final ACL configuration must conform to.

There are four primary nodes in the XML document, note that all four are optional.

5.1.1 Sections

<AdminServer>

This node of the XML document will defines a single ACLRule that will be applied as the Admin Server in the database ACL.

<CompulsoryEntries>

This node of the XML document will define entries that **must** appear in the final ACL (and what settings they must have). ACL entries specified in this section will be added if they are missing or updated if they are present so that their settings match those in the document.

<ForbiddenEntries>

This node of the XML document will define entries that **must not** appear in the final ACL. If present then these entries will be removed. If an entry in this section conflicts with an entry in the compulsory or permitted entries then this section takes precedence and the entry will be removed.

<PermittedEntries>

This node, if present, will specify entries (or more usually patterns of entries) that are **allowed** to be present in the final ACL configuration. All entries in the ACL will be removed except entries that match entries in this node or any entries in the **<CompulsoryEntries>** node. Entries that do match entries in this node may have their settings adjusted to conform to the entries or patterns specified here.

5.1.2 Entries

Individual ACL entries or groups of ACL entries matching a specified pattern are identified with an **<ACLRule>** node. An ACLRule node can optionally contains a number of **<Option>** nodes and/or a number of **<Role>** nodes. An Option node represents the setting on or off of a single ACL flag. A Role node associates the entry with membership or not of a specified ACL role.

5.1.2.1 ACLRule Node

The following attributes apply to an ACLRule Node.

5.1.2.1.1 Name Attribute

Either a Name attribute or a Pattern Attribute must be supplied on an ACLRule node. The name attribute specifies a string containing an abbreviated name that will be matched to an entry in the ACL list. If the name contains any “*” wildcards then these will be matched literally to the ACL entries, if you wish to do wildcard matching then specify a pattern attribute in place of the name attribute. If the name matches an entry in the ACL then the rest of the specification in the ACLRule will be applied to that entry. If the ACLRule entry is in the AdminServer section of the ACL Rule Stet then the name can specify a symbolic value of none, if this is specified then any entry in the ACL that is specified as the Admin Server will be changed to a regular entry.

5.1.2.1.2 Pattern Attribute

Either a Name attribute or a Pattern Attribute must be supplied on an ACLRule node. The pattern attribute specifies a string containing an abbreviated name with wildcards that will be matched to multiple entries in the ACL list. If the pattern matches any entries in the ACL then the rest of the specification in the ACLRule will be applied to those entries. ACLRule entries in the compulsory entries section of a rule set **CANNOT** specify patterns.

5.1.2.1.3 Type Attribute

The type attribute specifies the type that should be set in any matching ACL entries. The type should specify one of the following symbolic values.

- `Person` – A Person ACL entry.
- `Server` - A Server ACL entry.
- `PersonGroup` – A Person Group ACL entry.
- `ServerGroup` – A Server Group ACL entry.
- `MixedGroup` – A Mixed Group ACL entry.
- `Unspecified` – An Unspecified ACL entry.

5.1.2.1.4 Level Attribute

The level attribute specifies the access level that should be applied to any matching ACL entries. The level should specify one of the following symbolic values.

- `Manager` – Specified Manager access level.
- `Designer` – Specifies Designer access level.
- `Editor` – Specifies Editor access level.
- `Author` – Specifies Author access level.
- `Reader` – Specifies Reader access level.
- `Depositor` – Specifies Depositor access level.
- `NoAccess` – Specifies No Access .

5.1.2.2 Option Node

An Option node represents a single ACL flag and determines if the flag should be set on or off on any matching entries. The Option node has the following attributes.

5.1.2.2.1 Type Attribute

The type attribute identifies the specific ACL flag that should be set on or off. The type should specify one of the following symbolic values.

- `NoCreateDocs` – If set on the user cannot create documents in the database.
- `NoDeleteDocs` – If set on the user cannot delete documents in the database.
- `CreatePrivateAgents` – If set on then the user is permitted to create private agents in the database.

- `CreatePrivateFolders` – If set on then the user is permitted to create private views or folders in the database.
- `CreateFolders` – If set on then the user is permitted to create public folders in the database.
- `CreateLotusScript` – If set on then the user is permitted to create LotusScript agents in the database.
- `PublicReader` – If set on then the user has read access to pulic documents in the database.
- `PublicWriter` – If set on then the user has write access to pulic documents in the database.
- `NoMonitors` – If set on then the user cannot set monitors on the database.
- `NoReplicate` – If set on then the user cannot replicate or copy documents from the database.

5.1.2.2.2 Set Attribute

The set attribute specifies if the designated ACL flag should be set on or off. Specify one of the symbolic values of on or off for this attribute.

5.1.2.3 Role Node

The Role node is used to assign or un-assign a particular ACL role to any matching ACL entries. The Option node has the following attributes.

5.1.2.3.1 Name Attribute

The name attribute is used to identify the specific ACL Role that is to be assigned or un-assigned to the ACL entries. The value may be specified as * to assign or un-assign all of the ACL Roles defined in the database ACL.

5.1.2.3.2 Assign Attribute

The assign attribute determines if the specified role(s) should be assigned or un-assigned to any matching ACL entries. Specify one of the symbolic values yes or no.

5.1.3 Example Rule Sets

This section presents a number of example ACL rules XML documents that would be used to accomplish specific tasks in manipulating the ACL of databases.

5.1.3.1 Enforcing A Server Entry

The following XML document will make sure that there is a wildcard ACL entry for all servers in the infrastructure and that particular settings apply to the entry if it already exists.

```
<?xml version='1.0' encoding='utf-8'?>
<-- Add a generic server entry -->
<ACLRuleSet>
      <CompulsoryEntries>
            <ACLRule Name="*/SERVER/ACME" Type=ServerGroup Level=Manager>
```

```
                  <Option Type=NoDeleteDocs Set=Off/>
                  <Role Name=* Assign=Yes/>
            </ACLRule>
      </CompulsoryEntries>
</ACLRuleSet>
```

5.1.3.2 Removing Administrators from the ACL

The following XML document will remove any administrators that are listed explicitly in the ACL of any of the databases being processed.

```
<?xml version='1.0' encoding='utf-8'?>
<-- Remove explicitly listed administrators from the ACL -->
<ACLRuleSet>
      <ForbiddenEntries>
            <ACLRule Pattern="*/ADMIN/ACME>
            </ACLRule>
      </ForbiddenEntries>
</ACLRuleSet>
```

5.1.3.3 Preparing Databases for Archiving

This is a more complex example that achieves a number of adjustments to the ACLs of databases in preparation for archiving. Any Admin Server is removed. Archive administrative groups are added, general entries are corrected or added as necessary. Normal administrators are removed. Personal entries in the ACL are dropped to reader level access.

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE DXACL SYSTEM 'xmlschemas/domino_7_0.dtd'>
<ACLRuleSet>
      <AdminServer>
            <ACLRule Name=none Type=Server>
            </ACLRule>
      </AdminServer>
      <CompulsoryEntries>
            <ACLRule Name="*/SERVER/ACME" Type=ServerGroup Level=Manager>
                  <Option Type=NoDeleteDocs Set=Off/>
                  <Role Name=* Assign=Yes>
            </ACLRule>
            <ACLRule Name="LocalDomainServers" Type=ServerGroup
Level=Manager>
                  <Option Type=NoDeleteDocs Set=Off/>
                  <Role Name=* Assign=Yes>
            </ACLRule>
            <ACLRule Name="OtherDomainServers" Type=ServerGroup
Level=NoAccess>
            </ACLRule>
            <ACLRule Name="_4ArchiveAdministrators_Manager" Type=PersonGroup
Level=Manager>
                  <Option Type=NoDeleteDocs Set=Off/>
                  <Role Name=* Assign=Yes>
            </ACLRule>
            <ACLRule Name="-Default-" Type=Unspecified Level=NoAccess>
                  <Option Type=PublicReader Set=Off>
                  <Option Type=PublicWriter Set=Off>
            </ACLRule>
            <ACLRule Name="Anonymous" Type=Unspecified Level=NoAccess>
            </ACLRule>
            <ACLRule Name="_4ArchiveAccess_Users" Type=PersonGroup
Level=Reader>
            </ACLRule>
      </CompulsoryEntries>
      <ForbiddenEntries>
            <ACLRule Name="_4Operationsl_Admins"/>
      </ForbiddenEntries>
      <PermittedEntries>
            <ACLRule Pattern="*/*/ACME" Type=Person Level=Reader>
            </ACLRule>
      </PermittedEntries>
</ACLRuleSet>
```

5.2 Appendix B: 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.

5.2.1 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.

5.2.2 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>
```

5.2.2.1 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.

5.2.2.2 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.

5.2.2.3 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.

5.2.2.3.1 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.

5.2.2.3.2 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.

5.2.2.4 sign

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

5.2.3 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.

5.2.3.1 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.