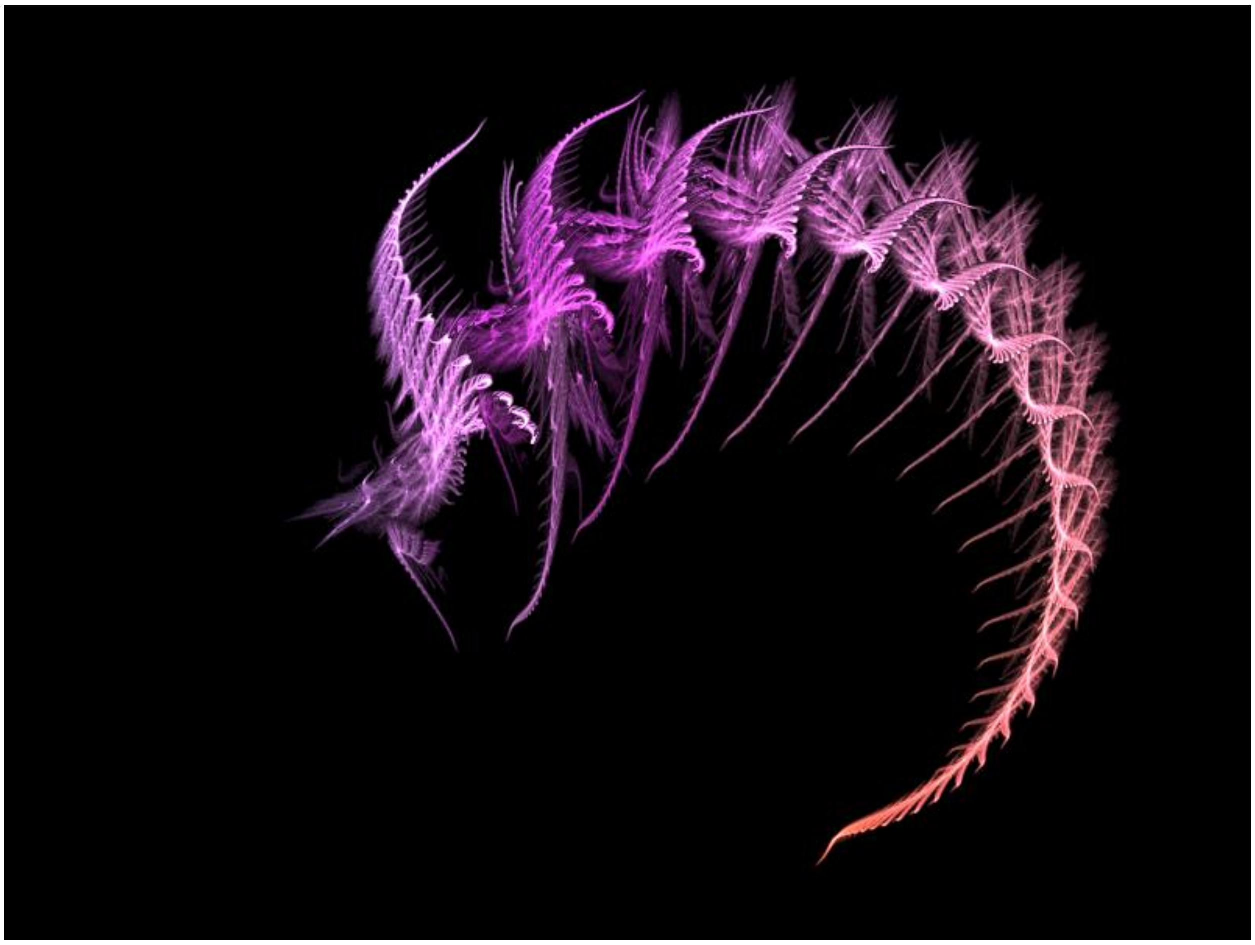
DX Tools

Using QCopy 2.6



Author: Ian Tree
Owner: HMNL b.*v.*
Customer: Public
Status: Draft
Date: 10/03/2012 12:04
Version: 2.6.1

Disposition: Open Source

Document History

Document Usage

Revision History

Date of this revision: 02/02/2015 12:04	Date of next revision *None*

Revision Number	Revision Date	Summary of Changes	Changes marked
2.5.0	11/02/11	Initial Base Version	No
2.5.3	28/02/12	QE Version	No
2.6.1	02/02/15	QE updates for version 2.6	No

Acknowlegements

Frontpiece Design was produced by the chaoscope application.

CHAOSCOPE.org
3D strange attractor rendering software

Contents

1. Introduction to QCopy

QCopy copies Notes databases, it is as simple and as complicated as that. The application is driven by a queue of transactions, each transaction is a request to copy a single database, or a collection of databases. Each copy operation is controlled through settings supplied in each transaction. The settings support many kinds of common operations that are involved in transformations of large volumes of databases. Transformations such as consolidations, mergers, de-mergers and all sorts of migrations can be addressed using QCopy.

QCopy is fast and resilient. Each individual database is copied by a multi-threaded copy engine and multiple databases can be copied in parallel allowing sustained high rates of data transfer.

The QCopy application is constructed as a Domino Server Add-In task, as such it is perfectly suited to unattended operations.

QCopy is a C++ application constructed on the DXCommon application kernel supporting Domino version from 6.0 through 9.0 on both Widows (32 & 64 bit) & Linux server platforms.

The copy engine at the heart of the QCopy application is mature and well tested component having been used in production environments for many years. It has been used in different configurations to copy over 300,000 databases with more than 200Tb of data.

The following options can be applied to individual copy operations.

- Adding an administrative group to the ACL.
- Disabling the “Enforce Consistent ACL” setting.
- Creating replica or non-replica copies.
- Copying private design elements.
- Copying deletion stubs.
- Copying Full-Text Index settings and building the Full-Text Index.
- Disabling agents in the copy.
- Building all view indexes in the copy.
- Enabling transaction logging on the copy.
- Setting the design template name on the copy.
- Refreshing the design of the copy.
- Running an agent in the copy of the database.
- Performing a final replication between the copy and the original databases.
- Converting Attachments to use LZ1 compression.
- Setting the ODS level of the target database.
- Enabling DAOS attachment storage on the target database.
- Selecting local encryption of the target database.

2. QCopy Transactions

This section of the document identifies the different options that can be used on a QCopy transaction and how these options should be used. Transactions can be manually added to the QCopy Control database or added programmatically. There is an action “New Transaction” visible at the top of the “To Be Processed” and the “To Be Approved” views in the database that will open the form to create a new transaction, or from the menu select “Create” and “QCopy Transaction”.

2.1 Pre-Processing Options

These options invoke processing that happens before the copy operation begins.

2.1.1 Delete Database

This option will delete the database that is the target of the copy operation. If the database does not exist then the transaction is not failed but continues with the copy step.

2.2 Source and Target Options

These options are used to select and optionally verify the source and target database.

Note: The specifications for the selection and copying of collections of databases (“Feeder Transactions”) are dealt with in a later section of this document.

2.2.1 Source Server

Supply the abbreviated name of the server from which you want to copy the database. A value of “Local” can be used to indicate the Local server i.e. the server on which the QCopy Add-In task is running. This parameter is mandatory.

2.2.2 Source Database

Supply the name of the source database that is to be copied, the name should be supplied with the path relative to the Notes Data Directory on the source server. This parameter is mandatory.

2.2.3 Source ReplicaID

This parameter is optional, if supplied then when the source database is opened the ReplicaID supplied is checked against the ReplicaID of the source database. If the ReplicaIDs do not match then the transaction is failed.

2.2.4 Target Server

Supply the abbreviated name of the server to which you want to copy the database. A value of “Local” can be used to indicate the Local server i.e. the server on which the QCopy Add-In task is running. This parameter is mandatory.

2.2.5 Target Database

Supply the name of the target database that is to be copied to, the name should be supplied with the path relative to the Notes Data Directory on the source server. This parameter is mandatory.

2.2.6 Create Replica Instead of Copy

This parameter is mandatory but defaults to “Copy”. If set to “Replica” then the target database is created as a replica of the source database. When set to “Copy” the target database is a non-replica copy of the source database.

2.2.7 Perform One-Time Replication

This parameter is mandatory but defaults to “No”. If set to “Yes” then after all copy operations and post-copy operations are completed, a one-time replication will be performed with the source.

This operation is useful in circumstances where the source and target replicas will continue to be replicated for some time after to copy process.

2.3 Copy Options

These options control the way in which the copy operation is performed.

2.3.1 Fixup Source Database

This parameter is mandatory but defaults to “No”. If set to “Yes” then while opening the source database a “database fixup” will be performed on the source database. It is comparatively rare to meet a situation where this option would be routinely employed. See the section on “Error Recovery Options” for more options for dealing with corrupt or damaged databases.

2.3.2 Admin Group

This parameter is optional. Supply the name of a Group that will be added to the ACL of the target database during the copy operation. The name provided will be added as a Manager with all permissions and Roles.

This option is typically used when moving databases between administrative domains such as during merger or de-merger operations. In this type of operation the database is typically pulled to a “stepping stone” server from a foreign administrative domain, this setting allows administrative control of the database to be assumed by the local administration. Additional ACL and other conditioning can be applied to the database on the “stepping stone” server before deploying it to its final destination.

2.3.3 Disable Consistent ACL

This parameter is mandatory but defaults to “No”. If set to “Yes” then if the source database has the “Enforce Consistent ACL” setting applied the setting is removed on the target database.

Having this setting on during migration and transformation processing can have adverse effects preventing successful replication of the database.

2.3.4 Copy Private Design Elements

This parameter is mandatory but defaults to “No”. If set to “Yes” then any private design elements that are present in the source database will be copied to the target database.

2.3.5 Copy Deletion Stubs

This parameter is mandatory but defaults to “No’. If set to “Yes” then any deletion stubs that are present in the source database will be copied to the target database.

If the target database will be replicated with an existing replica after the copy operation then this option should be set to “Yes”.

2.3.6 Disable Agents in Target

This parameter is mandatory but defaults to “No”. If set to “Yes” then the target database is marked as having no background agents, this setting has the effect of preventing the Agent Manager from scanning the target database to find agents that should be added to the agent schedule. There are many scenarios where the use of this option is appropriate. Note that this setting prevents the Agent Manager from scanning the target database for agents rather than disabling individual agents. This is therefore particularly appropriate for reducing the load on a server that will hold many databases that would not have any agents that should be run, for example, an archive server.

2.3.7 Copy Full-Text Index Settings

This parameter is mandatory but defaults to “No”. If set to “Yes” then any Full-Text Index settings that are present on the source database will be copied over to the target database.

2.3.8 Enable Transaction Logging

This parameter is mandatory but defaults to “No”. If set to “Yes” then the target database will be enabled for transaction logging when it is created.

2.4 Advanced Database Options

These options enable some advanced capabilities to be enabled on the target database.

2.4.1 ODS Level

This parameter allows selection of the ODS level to be used on the target database. This setting would normally only be used to back-level the ODS version to an older level than the default for the target server or to enforce a higher optional level that is supported by the server.

The following specific ODS levels are currently supported.

- R6.0 latest level (ODS: 43)
- R8.0 latest level (ODS: 48)
- R8.5 latest level (ODS: 51)
- R9.0 latest level (ODS: 53)

2.4.2 Local Encryption

This parameter controls the strength of local (on disk) encryption of the target database and may be set to one of the following “None” (default), “Simple”, “Medium” or “Complex”.

2.4.3 LZ1 Compression

Selecting “Yes” sets the default compression scheme for the target database to use the Lempel-Ziv-Walsh compression scheme as the default for the target database. New attachments made in the target database will be compressed using the LZ1 compression method. If this option is not selected then the target database will use the Huffman compression scheme for attachments as a default.

2.4.4 LZ1 Compress Existing

If the LZ1 compression scheme has been selected as the default for the target database then setting this option to “Yes” will cause existing attachments in the database to be recompressed using the LZ1 scheme. Attachments will not be compressed to the LZ1 scheme under the following conditions.

- The compressed file has one of the following file extensions .ZIP, .GZIP, .TAR, .GZ, .TGZ, .ARC, .CAB, .JAR, .ZIPX, .RAR.
- Compressing the attachment would make the compressed attachment larger than it is in the source database.

2.4.5 DAOS

Setting this option to “Yes” enables the use of DAOS attachment storage for the target database, providing that the DAOS service is enabled on the target server. All existing attachments in the database will be migrated to the DAOS attachment storage during the copy process. It is important to ensure that DAOS has been correctly enabled on the target server and that there is sufficient disk space allocated to the DAOS store before using this option for a copy operation.

2.5 Post Copy Options

These options control additional processing that will be applied to the target database after the main copy operation has been completed.

2.5.1 Build View Indexes

This parameter is mandatory but defaults to “No”. If set to “Yes” then all of the views in the target databases will be built.

2.5.2 Build Full-Text Index

This parameter is mandatory but defaults to “No”. If set to “Yes” then it causes the Full-Text Index to be built after the copying of all of the documents has completed.

2.5.3 Maintain Folder Content

This parameter is mandatory but defaults to “No” and only applies to non-replica copies, replica copies of a database will always have folder content updated in the target to reflect the source. If set to “Yes” then folders in the target database will be populated with the same content as is present in the source database.

2.5.4 Refresh Design

This parameter is mandatory but defaults to “No”. If set to “Yes” then the design of the database will be refreshed after the copy steps have completed. The refresh is performed with the template identified in the database and is carried out on the target server. The template name can be changed with the optional “Use Design Template” parameter.

Performing a design refresh after copying the source database to the target is not only useful for applying local standards but can also be used to apply local branding and for injecting code that will be used to condition the data for the new environment.

2.5.5 Use Design Template

This parameter is optional, if set then the name provided will be set as the template name for the target database to inherit its design from. This setting can be used with the "Refresh Design" setting to force design conformance as a integrated step in the copy process.

2.5.6 Run Agent after Copying

This parameter is optional. If specified then it should contain the name of an agent that is present in the design of the target database, either as copied from the source or as applied by the "Design Refresh" process. The named agent will be run after all other operations are completed.

Typical uses of this parameter would be to run a "Conditioning" agent that would updated the Readers and Authors fields in all documents in the database to conform to local access policy.

2.6 Error Recovery Options

These options will determine how the copy engine will respond to particular errors.

2.6.1 Allow Retries

This parameter is mandatory but defaults to "No". If set to "Yes" then in the case of an unrecoverable error condition the transaction will be reset and tried again at a later time.

2.6.2 Maximum Number of Retries

If the "Allow Retries" is set to "Yes" then this parameter determines how many time the transaction will be re-tried before being flagged as a permanent error.

2.6.3 Restart Options

If the "Allow Retries" option is selected then these settings determine what actions will be taken before the re-tried copy operation is attempted.

2.6.3.1 Delete Database

This setting causes the target database to be deleted before attempting to copy operation.

2.6.3.2 Fixup Source

This setting will cause a fixup to be performed on the source database before starting the copy operation. In certain cases this setting may be automatically set by error detection mechanisms in the copy engine.

2.6.4 Fault Tolerance Level

This section lists a series of fault "classes" that will be tolerated by the copy engine and will **not** result in an error condition.

2.6.4.1 Folder Errors

If faults are detected while populating the contents of folders and there faults cannot be fixed on-the-fly then this setting will prevent the copy engine from treating this condition as a permanent error.

2.6.4.2 Access Errors

If the copy engine is unable to read documents in the source database because it does not have reader access to them then this setting will prevent the copy engine from treating this condition as a permanent error. Refer to the section “Comparing Pull and Push Copy Operations” later in this document for more discussions on document access issues.

2.6.4.3 Other Copy Errors

If the copy engine is unable to read documents from the source database for any reason other than an “Access Error” or is unable to write a document to the target database then this setting will prevent it from being treated as a permanent error.

2.6.4.4 Resource Errors

If the copy engine is unable to complete copy operations due to a lack of resources on either the remote or the local server then this setting will prevent it from being treated as a permanent error.

2.7 Generic Transaction Options

The following settings do not affect the copy operation itself but are related to the logistics of executing the transaction itself.

2.7.1 Status

This parameter determines the current state of the transaction in the processing cycle. Refer to the section “Transaction Workflow” for more details.

2.7.2 Transaction Request ID

This parameter assigns a unique transaction identifier to each transaction that passes through the processing cycle. For manually created transactions this is set to a value computed by the @Unique formula language function. Refer to the section on “Programmatic Interfaces” for more details on how this item is used. Refer to the section on “Feeder Transactions” for more details on automatically generated transaction request IDs.

2.7.3 Approval Status

Transactions that are in the “NEW” transaction state are not available for processing until the approval status is set to “Approved”. This provides a simple workflow which can be useful when dealing with source and target servers that exist in different administrative or organisational domains.

Transactions that require approval are held in a separate queue and can be selected and approved “in bulk”.

2.7.4 Urgent Flag

Transactions can be selected and marked as being urgent, this moves the transactions to the head of the queue of transactions that are ready for execution.

2.8 Scheduling Options

Transactions may be scheduled for repeated or one-off execution. The following parameters are used to control individual transaction scheduling.

2.8.1 First Run Time

Supply the date and time when the transaction should be run for the first or only time.

2.8.2 Repeat

Specify when the transaction should be repeated. A value of “Never” causes the transaction to be run only a single time. “Daily”, “Weekly” or “Monthly” indicate that after the initial run the transaction will be run again at the indicated interval. The time supplied in the “First Run Time” setting will be the target time of day when the transaction will run again.

3. Feeder Transactions

3.1 General

"Feeder" transactions are single transactions that can be submitted to QCopy for processing that apply to a collection of databases rather than a single database. The simplest example of a collection of databases is a directory. Rather than having to supply an individual transaction for each database in a directory you can instead supply just the directory name. QCopy will recognise such a transaction and process each database in the directory as if a single database had been specified in the transaction.

3.1.1 Source Selection

The largest practical unit of source selection that you can apply is a Server. Specify an asterisk (*) in the source database field on a transaction to select every database on the source server for copying. There are not many situations that lend themselves to copying complete servers, taking snapshots of test servers for use in regression testing is about the only valid situation that we have come across in the wild.

The next unit of source selection that can be applied is multiple directories that match a pattern. Specify a directory name containing wildcard characters in the source database field on a transaction to select every directory within the Notes Data Directory that matches the pattern and every database within those directories that are selected. For example supplying a value of "mail?" for the source database would copy all databases in the "mail1", "mail2", "mail3" etc. directories. The selection of source databases will recurse into the sub-directories of the selected directories.

Specifying a directory name without any wildcards would result in all of the databases within the directory and any sub-directories and their databases to be selected for copying.

Further refinement can be made to selection, for instance, specifying "mail1\a*.nsf" would select all of the databases that start with the letter "a" from the "mail1" directory.

3.1.2 Target Database

The target database name is derived from the source database name located and the values that were specified for the source search pattern and the target database pattern. If "mail1" was specified for the source selection patter and "archive/mail1" was specified for the target name specification then the database mail1\abc.nsf" would by copied to a target of "archive/mail1/abc.nsf". The fixed portion of the source specification is replaced with the fixed portion of the target database name string.

3.1.3 Copy and Other Options

All options specified on the Feeder transaction will be used for each of the individual database copy operations.

3.1.4 Method of Operation

When a Feeder transaction is executed then each database that matches the source selection is located and the name for the target database is synthesised a new QCopy transaction is built, copying all of the options from the Feeder transaction. The new transaction is saved in the control database and will make its way up the ready queue until it is executed. Database copy transactions that are generated from a "Feeder" transaction are marked as urgent so that they will execute soon after the "Feeder" transaction that generated them and before any other "Feeder" transactions that are on the ready queue.

4. Transaction Workflow

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

4.1 New Transaction

When a transaction is created it is assigned a status of “NEW”, assuming that the approval flag has taken the default setting of “Not Approved” then the transition will sit in the “To Be Approved” queue. The transaction will remain in this queue indefinitely until it is approved. Multiple transactions can be selected in the “To Be Approved” queue and approved using the action at the top of the view.

4.2 Approved Transaction

When a NEW transaction is approved it is moved to the “Ready” queue. This queue is maintained in the order that transactions are created, however the Urgent Flag can be used to move individual transactions to the head of the queue.

The “Ready” queue is continuously monitored by the QCopy server add-in task, as soon as the add-in is in a state where it can run another transaction it will select the transaction at the head of the queue and start to execute it.

4.3 Server Availability Checks

When a transaction starts to be executed it is subjected to a server availability check. This consists of firstly checking if the remote server is up and running assuming that it is running then the number of users currently on both the source and target servers are checked against any limits that may have been specified in a server document in the control database. If the number of users exceeds the threshold for either source or target or the server is down then the transaction is marked as “DELAYED” and moved to the delayed queue.

4.4 In Progress Transaction

As soon as the QCopy server add-in task starts to execute a transaction then it is marked “IN PROGRESS”.

Should the server add-in task or indeed the server fail catastrophically then any transactions that were executing at the time of the failure (i.e. marked “IN PROGRESS”) will be automatically restarted.

The transaction will remain in the “IN PROGRESS” state until it completes or fails.

4.5 Completed Transaction

When a transaction completes all process steps without any faults being detected it is changed to the “COMPLETED” state. This is a terminal state the transaction makes no further transitions.

4.6 Error Transaction

If the transaction experienced some kind of fault during execution and the “Allow Retries” option is **not** set to “Yes” on the transaction then the transaction is marked as an “ERROR” and moved to the errors queue. This is a terminal state the transaction makes no further transitions.

4.7 Retried Transaction

If the transaction experienced some kind of fault during execution and the “Allow Retries” option is set and the retry count for the transaction has not exceeded the Maximum Number of Retries then the transaction will be retried. One copy of the failed transaction has its state set to “RETRIED” and moved to the retried queue, this copy of the transaction holds the log and processing information from the failed execution attempt. A second fresh copy is put on the delayed queue.

4.8 Delayed Transaction

If a transaction fails a server availability check or experienced a fault but was eligible to be retried then it is marked as “DELAYED” and moved to the delayed transaction queue. The QCopy server add-in tasks monitors the delayed transaction queue any time that it has no work available on the transaction ready queue and will execute transactions from that queue providing that they have been on the queue for a minimum of 15 minutes.

5. Programmatic Interface

It is a relatively trivial task to interface the QCopy control databases to other applications. There are minimal requirements for creating a new transaction and a view is provided for monitoring the progress of individual transactions.

5.1 Creating New Transactions

The following fields should be set as indicated as a minimum to form a valid QCopy transaction.

- Form – "QCTX".
- Status – "NEW".
- Approved – "Yes" unless you require the transaction to be manually approved in the control database before it is executed in which case set it to "No".
- TransReqID – This is the transaction ID and should be set to a unique value on each transaction. **Hint**, you can set this to the UNID (@DocumentUniquID) of the source transaction to provide an easy reference between the application and the control database.
- SourceServer – Set this field to the abbreviated name of the server from which database(s) will be copied, set the field to "Local" to copy from the current server.
- SourceDatabase – The name of the database to be copied.
- TargetServer – Set this field to the abbreviated name of the server to which database(s) will be copied, set the field to "Local" to copy to the current server.
- SourceDatabase – The name of the copied database.

The following fields can optionally be set to change the processing options of the transaction.

- Urgent – set the value to "Yes" or "No" to affect the urgency of the transactions.
- AllowRetry – set the value to "1" to permit the transaction to be retried if it fails.
- RetryLimit – set this value to a string containing the number of times that the transactions can be retried.
- PreProcessActions – set this value to a string containing the sum of any required actions
  - 1 – Delete the target database before processing.
  - 2 – Perform a fixup on the source database before processing.
- AdminGroup – Set this field to a Group name that will be added to the ACL of the target database.
- UnenforceConsistentACL – Set this field to "1" to turn off the enforce consistent ACL in the target database.
- MakeReplica – Set this field to "1" to make the target database a replica copy of the source database.
- CopyPrivateDesign – Set this field to "1" to copy any private design elements in the source database into the target database.
- CopyStubs – Set this field to "1" to copy deletion stubs from the source to the target database.
- CopyFTISpec – Set this field to "1" to copy the Full-Text Index settings from the source to the target database.
- BuildFTI – Set this field to "1" to build the Full-Text Index on the target database.
- DisableAgents – Set this field to "1" to disable background agents in the target database.

- RefreshDesign – Set this field to “1” to force a refresh of the design of the target database.
- BuildIndexes – Set this field to “1” to build the indexes of all views in the target database.
- EnableTXLogging – Set this field to “1” to enable transaction logging on the target database.
- CopyFolders – Set this field to “1” to copy the folder contents in a non-replica copy operation.
- UpdateTemplate – Set this field to a database design template name to be applied to the target database.
- ConditionAgent – Set this field to the name of an agent to be run in the target database after all copy processing has completed.
- OneTimeRep – Set this field to “1” to force a one-time replication between the source and target after copying has completed.
- ODSLevel – Set the field to “0” to use the default level on the target server. The field can also be set to the following values:
  - “60” - For the R6.0 latest level (ODS: 43)
  - “80” - For the R8.0 latest level (ODS: 48)
  - “85” - For the R8.5 latest level (ODS: 51)
  - “90” - For the R9.0 latest level (ODS: 53)
- LocalEncryption – Set the field to “0” if the target database is not to be encrypted. The field can also be set to the following values:
  - “1” - Encrypt the target database with “simple” level of encryption
  - “2” - Encrypt the target database with “medium” level of encryption
  - “3” - Encrypt the target database with “complex” level of encryption
- LZ1Compress – Set the field to “1” to force the default compression for attachments to be LZ1.
- LZ1CompressExisting – Set the field to “1” to have existing attachments in the database recompressed using the LZ1 scheme.
- DAOS – Set this field to “1” to enable DAOS attachment storage on the target database. As this option is set at the start of the copy process this will cause existing attachments to be migrated from the database into the DAOS store. You must of course have enabled DAOS storage on the target server before using this option.

5.2 Monitoring Transaction Progress

Use the “LookupByTransactionID” view in the database to track the progress of individual transactions. The view is keyed on the Transaction ID. Applications should normally check the “Status” field on an individual transaction to determine the current state. The following states are regarded as being terminal and should be acted on by the application.

- “COMPLETED” – the transaction has been executed and has completed processing without any errors.
- “ERROR” – the transaction has failed, it may have been retried a number of times before reaching this state.

All other values of the transaction status field should be considered as transient and should not be acted on by the application.

If permitted and a transaction fails and is retried then there will be more than one transaction document in the control database with the same transaction ID. Refer to the “Transaction Workflow” section of this document for details.

6. Installing QCopy

6.1 Building the QCopy Server Add-In Task

The QCopy server add-in task is built on the Domino eXplorer Tools DXCommon kernel and the Notes C API, you need to download and install these before you can build the QCopy application.

6.1.1 Reference Environments

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

Windows:

Build Environment:

Microsoft Visual Studio 2005/2008/2010/2011

Running on any supported windows workstation.

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

Notes API Version 9.0

Execution Environment:

Windows Standard Server 2008 R2 (32 bit or 64 bit)

Domino Server 9.0.1 FP1 (32 and 64 bit).

Note: Execution environments from Domino 6.5.x through 9.0.x are regularly used.

Linux:

Build Environment:

Gcc Version: 4.1.2 for i386-redhat-linux.

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

Notes API Version 8.5

Execution Environment:

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

Domino Server 8.5.1 FP3.

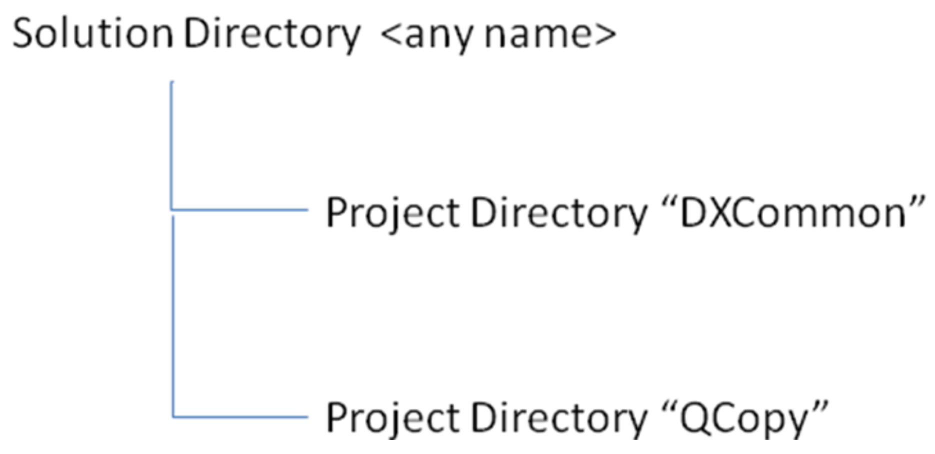
Note: Execution environments from Domino 7.0.x through 9.0.x are regularly used.

6.1.2 Notes API Installation

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

6.1.3 Directory Structure

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


Solution Directory <any name>
Project Directory "DXCommon"
Project Directory "QCopy"

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

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

6.1.4 Installing the DXCommon Kernel Sources

Windows:

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

As an example.

Unpack the DXCommon kernel into a directory "c:\usr\include\DXCommon-3.14.0" and then create the symbolic link from within the solution directory using the following command.

```
mklink /D DXCommon "c:\usr\include\DXcommon-3.14.0"
```

Linux:

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

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

As an example.

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

```
ln -s /usr/include/DXCommon-3.14.0 DXCommon
```

6.1.5 Installing the QCopy Sources

Windows:

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

Header Files

AppCommandHandler.h

AppRunSettings.h

AppTransactionHandler.h

QCFeeder.h

QCFeedProcessor.h

QCopier.h

QCopy.h

QCopyRequest.h

QFeedRequest.h

Source Files

AppCommandHandler.cpp

AppRunSettings.cpp

AppTransactionHandler.cpp

QCFeeder.cpp

QCFeedProcessor.cpp

QCopier.cpp

QCopy.cpp

QCopyRequest.cpp

QFeedRequest.cpp

Linux:

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

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

6.1.6 Build Settings

Windows:

Add each source and header file that is used from the DXCommon kernel to the QCopy project. It is convenient to add these source and header files into subsets that can then be copied into other projects, in Visual Studio 20xx these collections are referred to as “filters”. To create a filter right-click on either the “Header Files” folder or the “Source Files” folder and select “Add” then “New Filter”. The following filters are convenient to use in the QCopy project “Copier”, “Explorer” and “Runtime” these filters should be created in both the “Source Files” and “Header Files” folders. To populate the individual filter right-click on the filter then select “Add” then “Existing Item” navigate to the required source or header file(s), select the file(s) and click the “Add” button. The contents of each filter are listed below.

Header Files\Copier

DXCommon\DBC\CopyRequest.h

DXCommon\DBC\DbCopier.h

DXCommon\DBC\PartCopyrequest.h

Header Files\Explorer

DXCommon\MTDX\DominoExplorer.h

DXCommon\MTDX\DXDbScanner.h

DXCommon\MTDX\DXDirScanner.h

DXCommon\MTDX\DXFilter.h

DXCOmmon\MTDX\DXDXReporter.h

DXCommon\MTDX\DXRequest.h

DXCommon\MTDX\DXServerScanner.h

DXCommon\MTDX\DXSpider.h

DXCommon\MTDX\DXSStash.h

Header Files\Runtime

DXCommon\APIPackages.h

DXCommon\MTX\CommandHandler.h

DXCommon\DXException.h

DXCommon\DXGlobals.h

DXCommon\ElapsedTimer.h

DXCommon\Debug\Helper.h

DXCommon\MTX\MTExecutive.h

DXCommon\Platform\NotesBase.h

DXCommon\Platform\PlatBase.h

DXCommon\Threads\Runnable.h

DXCommon\RunSettings.h

DXCommon\Threads\ThreadDispatcher.h

DXCommon\Threads\ThreadManager.h

DXCommon\Threads\ThreadManagerPolicy.h

DXCommon\Threads\ThreadMonitor.h

DXCommon\Threads\ThreadScheduler.h

DXCommon\Threads\ThreadStructs.h

DXCommon\MTX\TransactionHandler.h

DXCommon\MTX\TransactionQueue.h

DXCommon\Threads\WorkerThread.h

Source Files\Copier

DXCommon\DBC\CopyRequest.cpp

DXCommon\DBC\DbCopier.cpp

DXCommon\DBC\PartCopyrequest.cpp

Source Files\Explorer

DXCommon\MTDX\DominoExplorer.cpp

DXCommon\MTDX\DXDbScanner.cpp

DXCommon\MTDX\DXDirScanner.cpp

DXCommon\MTDX\DXFilter.cpp

DXCOmmon\MTDX\DXDXReporter.cpp

DXCommon\MTDX\DXRequest.cpp

DXCommon\MTDX\DXServerScanner.cpp

DXCommon\MTDX\DXSpider.cpp

DXCommon\MTDX\DXSStash.cpp

Source Files\Runtime

DXCommon\APIPackages.cpp

DXCommon\MTX\CommandHandler.cpp

DXCommon\DXException.cpp

DXCommon\ElapsedTimer.cpp

DXCommon\Debug\Helper.cpp

DXCommon\MTX\MTExecutive.cpp

DXCommon\Threads\Runnable.cpp

DXCommon\RunSettings.cpp

DXCommon\Threads\ThreadDispatcher.cpp

DXCommon\Threads\ThreadManager.cpp

DXCommon\Threads\ThreadManagerPolicy.cpp

DXCommon\Threads\ThreadMonitor.cpp

DXCommon\Threads\ThreadScheduler.cpp

DXCommon\MTX\TransactionHandler.cpp

DXCommon\MTX\TransactionQueue.cpp

DXCommon\Threads\WorkerThread.cpp

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

For Windows 32 bit

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

For Windows 64 bit

Section/Entry	Release Setting	Debug Setting
General		
Character Set	Not Set	Not Set
C/C++		
Preprocessor		
Preprocessor Definitions	WIN32;NDEBUG;_CONSOLE;NT;W32;W;W64;ND64;_AMD64_;ND64SERVER	WIN32;_DEBUG;_CONSOLE;NT;W32;W;W64;ND64;_AMD64_;ND64SERVER
Code Generation		
Runtime Library	Multi-threaded (/MT)	Multi-threaded Debug (/MTd)
Struct Member Alignment	1 Byte (/Zp1)	**Default**
Command Line		
Additional Options	/Oy-	/Oy-
Linker		
Input		
Additional Dependencies	notes.lib	notes.lib Dbghelp.lib Psapi.lib

Notes:

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

/Zp1 packing is a Notes API requirement as all Notes API structures are packed and not padded or member aligned. **IMPORTANT**: Do not set this option for Windows x64 (64 bit) builds, Notes uses the default structure packing on Windows 64 bit.

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

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

Linux:

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

make QCopy

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

make rebuild QCopy

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

make rebuild QCopy BV=DBG

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

```
#
#  Build for QCopy 2.6.1 -- Build: 109 -- DXCommon 3.14.0
#
#  Optional targets:
#
#  rebuild     -  force a complete rebuild of the target
#
#  macros:
#
#  BV=DBG      -  Builds the DEBUG variant of the target
#

TARGET = QCopy

#  Define the primary source files

SOURCES = $(TARGET).cpp
SOURCES += AppRunSettings.cpp
SOURCES += AppCommandHandler.cpp
SOURCES += AppTransactionHandler.cpp
SOURCES += QCFeeder.cpp
SOURCES += QCFeedProcessor.cpp
SOURCES += QCopier.cpp
SOURCES += QCopyRequest.cpp
SOURCES += QFeedRequest.cpp
HEADERS = $(TARGET).h
HEADERS += AppRunSettings.h
HEADERS += AppCommandHandler.h
HEADERS += AppTransactionHandler.h
HEADERS += QCFeeder.h
HEADERS += QCFeedProcessor.h
HEADERS += QCopier.h
HEADERS += QCopyRequest.h
HEADERS += QFeedRequest.h

#  Define the D/B Copier modules to build

COPY_SOURCES = ../DXCommon/DBC/Attachments.cpp
COPY_SOURCES += ../DXCommon/DBC/CopyRequest.cpp
COPY_SOURCES += ../DXCommon/DBC/DbCopier.cpp
COPY_SOURCES += ../DXCommon/DBC/PartCopyRequest.cpp
COPY_HEADERS = ../DXCommon/DBC/Attachments.h
COPY_HEADERS += ../DXCommon/DBC/CopyRequest.h
COPY_HEADERS += ../DXCommon/DBC/DbCopier.h
COPY_HEADERS += ../DXCommon/DBC/PartCopyRequest.h

#  Define Core modules to build

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

#  Define the Multi-Threaded Core modules to build

MTCORE_SOURCES += ../DXCommon/MTX/CommandHandler.cpp
MTCORE_SOURCES += ../DXCommon/MTX/MTExecutive.cpp
MTCORE_SOURCES += ../DXCommon/MTX/TransactionHandler.cpp
MTCORE_SOURCES += ../DXCommon/MTX/TransactionQueue.cpp
MTCORE_SOURCES += ../DXCommon/Threads/ThreadManagerPolicy.cpp
MTCORE_SOURCES += ../DXCommon/Threads/ThreadManager.cpp
MTCORE_SOURCES += ../DXCommon/Threads/ThreadScheduler.cpp
MTCORE_SOURCES += ../DXCommon/Threads/ThreadDispatcher.cpp
MTCORE_SOURCES += ../DXCommon/Threads/ThreadMonitor.cpp
MTCORE_SOURCES += ../DXCommon/Threads/WorkerThread.cpp
MTCORE_SOURCES += ../DXCommon/Threads/Runnable.cpp
MTCORE_HEADERS = ../DXCommon/MTX/CommandHandler.h
MTCORE_HEADERS += ../DXCommon/MTX/MTExecutive.h
MTCORE_HEADERS += ../DXCommon/MTX/TransactionHandler.h
MTCORE_HEADERS += ../DXCommon/MTX/TransactionQueue.h
MTCORE_HEADERS += ../DXCommon/Threads/ThreadManagerPolicy.h
MTCORE_HEADERS += ../DXCommon/Threads/ThreadManager.h
MTCORE_HEADERS += ../DXCommon/Threads/ThreadScheduler.h
MTCORE_HEADERS += ../DXCommon/Threads/ThreadDispatcher.h
MTCORE_HEADERS += ../DXCommon/Threads/ThreadMonitor.h
MTCORE_HEADERS += ../DXCommon/Threads/WorkerThread.h
MTCORE_HEADERS += ../DXCommon/Threads/Runnable.h

#  Define the Domino Explorer modules to build

EXPLORER_SOURCES = ../DXCommon/MTDX/DominoExplorer.cpp
EXPLORER_SOURCES += ../DXCommon/MTDX/DXDbScanner.cpp
EXPLORER_SOURCES += ../DXCommon/MTDX/DXDirScanner.cpp
EXPLORER_SOURCES += ../DXCommon/MTDX/DXFilter.cpp
EXPLORER_SOURCES += ../DXCommon/MTDX/DXReporter.cpp
EXPLORER_SOURCES += ../DXCommon/MTDX/DXRequest.cpp
EXPLORER_SOURCES += ../DXCommon/MTDX/DXServerScanner.cpp
EXPLORER_SOURCES += ../DXCommon/MTDX/DXSpider.cpp
EXPLORER_SOURCES += ../DXCommon/MTDX/DXSStash.cpp
EXPLORER_HEADERS = ../DXCommon/MTDX/DominoExplorer.h
EXPLORER_HEADERS += ../DXCommon/MTDX/DXDbScanner.h
EXPLORER_HEADERS += ../DXCommon/MTDX/DXDirScanner.h
EXPLORER_HEADERS += ../DXCommon/MTDX/DXFilter.h
EXPLORER_HEADERS += ../DXCommon/MTDX/DXReporter.h
EXPLORER_HEADERS += ../DXCommon/MTDX/DXRequest.h
EXPLORER_HEADERS += ../DXCommon/MTDX/DXServerScanner.h
EXPLORER_HEADERS += ../DXCommon/MTDX/DXSpider.h
EXPLORER_HEADERS += ../DXCommon/MTDX/DXSStash.h

#  Define the Object Lists

OBJECTS = $(TARGET).o
OBJECTS += AppRunSettings.o
OBJECTS += AppCommandHandler.o
OBJECTS += AppTransactionHandler.o
OBJECTS += QCFeeder.o
OBJECTS += QCFeedProcessor.o
OBJECTS += QCopier.o
OBJECTS += QCopyRequest.o
OBJECTS += QFeedRequest.o
COPY_OBJECTS = CopyRequest.o
COPY_OBJECTS += DbCopier.o
COPY_OBJECTS += PartCopyRequest.o
CORE_OBJECTS = APIPackages.o
CORE_OBJECTS += DXException.o
CORE_OBJECTS += ExecEnvironment.o
CORE_OBJECTS += ElapsedTimer.o
CORE_OBJECTS += RunSettings.o
MTCORE_OBJECTS = CommandHandler.o
MTCORE_OBJECTS += MTExecutive.o
MTCORE_OBJECTS += TransactionHandler.o
MTCORE_OBJECTS += TransactionQueue.o
MTCORE_OBJECTS += ThreadManagerPolicy.o
MTCORE_OBJECTS += ThreadManager.o
MTCORE_OBJECTS += ThreadScheduler.o
MTCORE_OBJECTS += ThreadDispatcher.o
MTCORE_OBJECTS += ThreadMonitor.o
MTCORE_OBJECTS += WorkerThread.o
MTCORE_OBJECTS += Runnable.o
EXPLORER_OBJECTS = DominoExplorer.o
EXPLORER_OBJECTS += DXDbScanner.o
EXPLORER_OBJECTS += DXDirScanner.o
EXPLORER_OBJECTS += DXFilter.o
EXPLORER_OBJECTS += DXReporter.o
EXPLORER_OBJECTS += DXRequest.o
EXPLORER_OBJECTS += DXServerScanner.o
EXPLORER_OBJECTS += DXSpider.o
EXPLORER_OBJECTS += DXSStash.o

#  Define the build options

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

#  Reles to build DEBUG or release targets

$(TARGET): $(OBJECTS) $(COPY_OBJECTS) $(CORE_OBJECTS) $(MTCORE_OBJECTS) $(EXPLORER_OBJECTS)
	$(CC) $(LINKOPTS) $(OBJECTS) $(COPY_OBJECTS) $(CORE_OBJECTS) $(MTCORE_OBJECTS)
$(EXPLORER_OBJECTS) -L$(NOTESDIR) -Wl,-rpath-link $(NOTESDIR) $(LIBS)
$(OBJECTS): $(SOURCES) $(HEADERS)
ifeq ($(BV), DBG)
	$(CC) $(CCOPTS) $(DEFINES) -D_DEBUG -I$(INCDIR) $(SOURCES)
else
	$(CC) $(CCOPTS) $(DEFINES) -I$(INCDIR) $(SOURCES)
endif
$(COPY_OBJECTS): $(COPY_SOURCES) $(COPY_HEADERS)
ifeq ($(BV), DBG)
	$(CC) $(CCOPTS) $(DEFINES) -D_DEBUG -I$(INCDIR) $(COPY_SOURCES)
else
	$(CC) $(CCOPTS) $(DEFINES) -I$(INCDIR) $(COPY_SOURCES)
endif
$(CORE_OBJECTS): $(CORE_SOURCES) $(CORE_HEADERS)
ifeq ($(BV), DBG)
	$(CC) $(CCOPTS) $(DEFINES) -D_DEBUG -I$(INCDIR) $(CORE_SOURCES)
else
	$(CC) $(CCOPTS) $(DEFINES) -I$(INCDIR) $(CORE_SOURCES)
endif
$(MTCORE_OBJECTS): $(MTCORE_SOURCES) $(MTCORE_HEADERS)
ifeq ($(BV), DBG)
	$(CC) $(CCOPTS) $(DEFINES) -D_DEBUG -I$(INCDIR) $(MTCORE_SOURCES)
else
	$(CC) $(CCOPTS) $(DEFINES) -I$(INCDIR) $(MTCORE_SOURCES)
endif
$(EXPLORER_OBJECTS): $(EXPLORER_SOURCES) $(EXPLORER_HEADERS)
ifeq ($(BV), DBG)
	$(CC) $(CCOPTS) $(DEFINES) -D_DEBUG -I$(INCDIR) $(EXPLORER_SOURCES)
else
	$(CC) $(CCOPTS) $(DEFINES) -I$(INCDIR) $(EXPLORER_SOURCES)
endif

#  Phony target for forcing a complete rebuild

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

Notes:

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

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

6.1.7 Building and Deploying the Application

Windows:

Select “Build” and then “Build QCopy”.

Copy the resulting executable (QCopy.exe) and the associated Program Debug Database (QCopy.pdb) to the Notes Executable directory on the server where you want to run the Add-In.

Linux:

From the QCopy project directory issue the “make QCopy” command or the “make rebuild QCopy” or the “make rebuild QCopy BV=DBG” according to the type of build that you require.

make QCopy

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

make rebuild QCopy

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

make rebuild QCopy BV=DBG

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

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

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

chown server qcopy

chgrp notes qcopy

chmod 2555 qcopy

6.2 Installing the QCopy Control Database

The installation of the control database is done from a “Virtual Template” that is available on the internet, this section assumes that you have available and installed the Remote Database Create (**Windows:** RDBCreate.exe **Linux:** rdbcreate) tool. If you do not have this tool then download the DXTool source for RDBCreate and build it. Refer to the “Using RDBCreate” manual.

6.2.1 *Install the Database*

From a command window go to the Notes Executable directory where RDBCreate exists enter the following command.

RDBCreate <server name> <database name> <templateURL> -V

Where:

<server name> is the abbreviated name of the server on which you want to install the control database.

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

<templateURL> is the URL for the Virtual Template you wish to install.

For the QCopy Control Database, use the following URL:

```
http://hmnl.nl/HMNL/DX/VTTDepot.nsf/Payloads/qcopy22.manifest/$File/manifest.xml
```

7. Starting QCopy on the Server

The QCopy application can now be loaded on the server where it was installed. From a remote console session with the server issue the following command.

“load QCopy <database name> <options>”

Where:

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

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

7.1.1 Command Line Options

The following command line options are available with QCopy.

Option	Meaning
-V	Sets the logging level to verbose. This provides more detailed logging messages to be written to the application log.
-T[:nn]	Sets the logging level to trace, This is a diagnostic setting that provides detailed logging messages to the application log. The optional “nn” setting restricts tracing to a designated are of the DXCommon kernel code. Refer to the DXGlobals.h header file for values that this setting can take.
-D[:nn]	Sets the logging level to debug, This is a diagnostic setting that provides even more detailed logging messages to the application log. The optional “nn” setting restricts tracing to a designated are of the DXCommon kernel code. Refer to the DXGlobals.h header file for values that this setting can take.
-E	Sets console echo mode on. Application log entries are written to documents in the control database. When console echo mode is on then all application log messages are echoed to the Domino console and the Domino system log database.
-M:nn	Sets the number of threads that the application will use. This defaults to 10 threads, refer to the section on “Usage Scenarios” for a discussion on using this setting.
-X:nn	Sets the number of transactions that can execute in parallel. This defaults to 1. Therefore, by default, QCopy transactions will execute serially.

8. QCopy Tell Commands

8.1 QCopy Message Queues

While it is running certain aspects of QCopy operation can be controlled by the use of “Tell” commands entered through the Domino console.

QCopy is capable of running multiple instances on the same Domino server, each instance will have a separate message queue that it monitors for commands. When an instance of QCopy is started it locales the lowest number available to use for the message queue name in the form “QCopy”<suffix> where <suffix> is a digit that starts at 1. Therefore the first, or only, instance of QCopy can be addressed in the form “Tell QCopy1 <command>”.

8.2 Commands

8.2.1 Quit

The quit command is not normally used, however, during a server shutdown the server issues this command on all message queues. QCopy will respond to the command by shutting down.

8.2.2 Stop [now]

The stop command is used to shut down QCopy in an orderly manner. Any transactions that are currently running will be completed, no new transactions are dispatched and the Server Add-In will shut down. Specifying the optional “now” parameter on the stop command causes QCopy to fail any transactions that are currently running and then shut down in an orderly manner.

8.2.3 Abort

The abort command is an alias for the “stop now” command.

8.2.4 Suspend

The suspend command tells QCopy to stop executing new transactions from the ready queue. Any transactions that are currently executing are completed, the Add-In task continues to run but will not process any transactions until the “resume” command is executed.

8.2.5 Resume

The resume command is the counterpart of the suspend command. The command only has any effect if the Add-In task is in the suspended state, then it causes the processor to resume processing transactions from the ready queue.

8.2.6 Verbose

The verbose command causes the logging mode of the processor to be switched to verbose mode, in this mode more detailed logging is made to the application log.

8.2.7 Loud

The loud command is an alias for the verbose command.

8.2.8 Terse

The terse command switches the logging mode of the processor back to normal mode, in this mode minimal logging is done to the application log.

8.2.9 Quiet

The quiet command is an alias for the terse command.

8.2.10 Echo [on|off]

The echo command without any parameters is the same as the "echo on" command it will cause all current application logging to be echoed to the Domino Server console and therefore the Domino log. The "echo off" command turns off the echo of application logging.

8.2.11 Noecho

The noecho command is an alias for "echo off".

8.2.12 Trace [nnn]

The trace command sets the processor logging functions into trace mode. The number on the command designates a particular are of function to be traced. Refer to the DXGlobals.h header file for the different trace area specifications.

This command should only be used for problem diagnosis.

In this mode very detailed logging is produced in the application log.

8.2.13 Debug [nnn]

The debug command sets the processor logging functions into debug mode. The number on the command designates a particular are of function to be traced. Refer to the DXGlobals.h header file for the different trace area specifications.

This command should only be used for problem diagnosis.

In this mode even more detailed logging is produced in the application log.

8.2.14 Refresh

The refresh command causes the QCopy processor to finish processing any transactions that are currently processing and then reset the processing environment to the default configuration and resume processing transactions.

8.2.15 Status

The status command causes the processor to display the current state of the processor and some volumetric information about how many transactions have been processed.

Sample output:

```
01/06/2011 14:17:11 CET: DXR0907I: Command received: status. [500]
01/06/2011 14:17:11 CET: QCP0201I: 1 transaction have been dispatched, 0 completed, 1 are
running, max concurrency is 1. [500]
01/06/2011 14:17:11 CET: QCP0208I: Transactions marked Completed: 0, Error: 0, Retried: 0,
Delayed: 0. [500]
```

8.2.16 Stats [thread|debug]

The stats command causes a number of current values of statistics from the DXCommon kernel to be written to the applications log. The thread parameter on the command adds certain additional “per thread” statistics to be output. The debug operand on the command causes the “per thread” statistics to be included along with more details. Understanding these statistics is beyond the scope of this document, refer to the documents about the architecture of the DXCommon kernel to gain insight into the meaning of these statistics.

8.2.17 Panic [message]

The panic command will trigger an NSD exception from within the processor. This is an extreme diagnostic aid as it will cause an NSD of the entire server. The optional message is recorded in the log and in the memory displayed in the NSD dump.

8.2.18 Maxtrans nn

The maxtrans command causes the processor to change the number of transactions that can be executed in parallel to be changed to the value supplied on the command.

8.3 Debugging Commands

The following commands are **ONLY** available in the debug compiled Windows version of the QCopy executable.

8.3.1 Memory

The memory command shows a report on current memory usage by the application, these statistics are reported to the server console and the application log.

Sample Output:

```
28/02/2012 14:05:16 CET: DXR0704I: Current(2) Working Set size: 128784 Kb, +98332 Kb since last
measured, +98336 Kb since first measured, Peak: 128784 Kb. [1]

28/02/2012 14:05:16 CET: DXR0704I: Current(2) Paged Pool use: 1410 Kb, +8 Kb since last measured,
+8 Kb since first measured, Peak: 1410 Kb. [1]

28/02/2012 14:05:16 CET: DXR0704I: Current(2) Non-Paged Pool use: 18 Kb, +7 Kb since last
measured, +7 Kb since first measured, Peak: 18 Kb. [1]

28/02/2012 14:05:16 CET: DXR0704I: Current(2) Normal Objects on the Heap: 32 , +5  since last
measured, +5  since first measured, Peak: 32 . [1]

28/02/2012 14:05:16 CET: DXR0704I: Current(2) Normal Objects Allocation: 148 Kb, +2 Kb since last
measured, +2 Kb since first measured, Peak: 148 Kb. [1]

28/02/2012 14:05:16 CET: DXR0704I: Current(2) Client Objects on the Heap: 0 , 0  since last
measured, 0  since first measured, Peak: 0 . [1]

28/02/2012 14:05:16 CET: DXR0704I: Current(2) Client Objects Allocation: 0 Kb, 0 Kb since last
measured, 0 Kb since first measured, Peak: 0 Kb. [1]
```

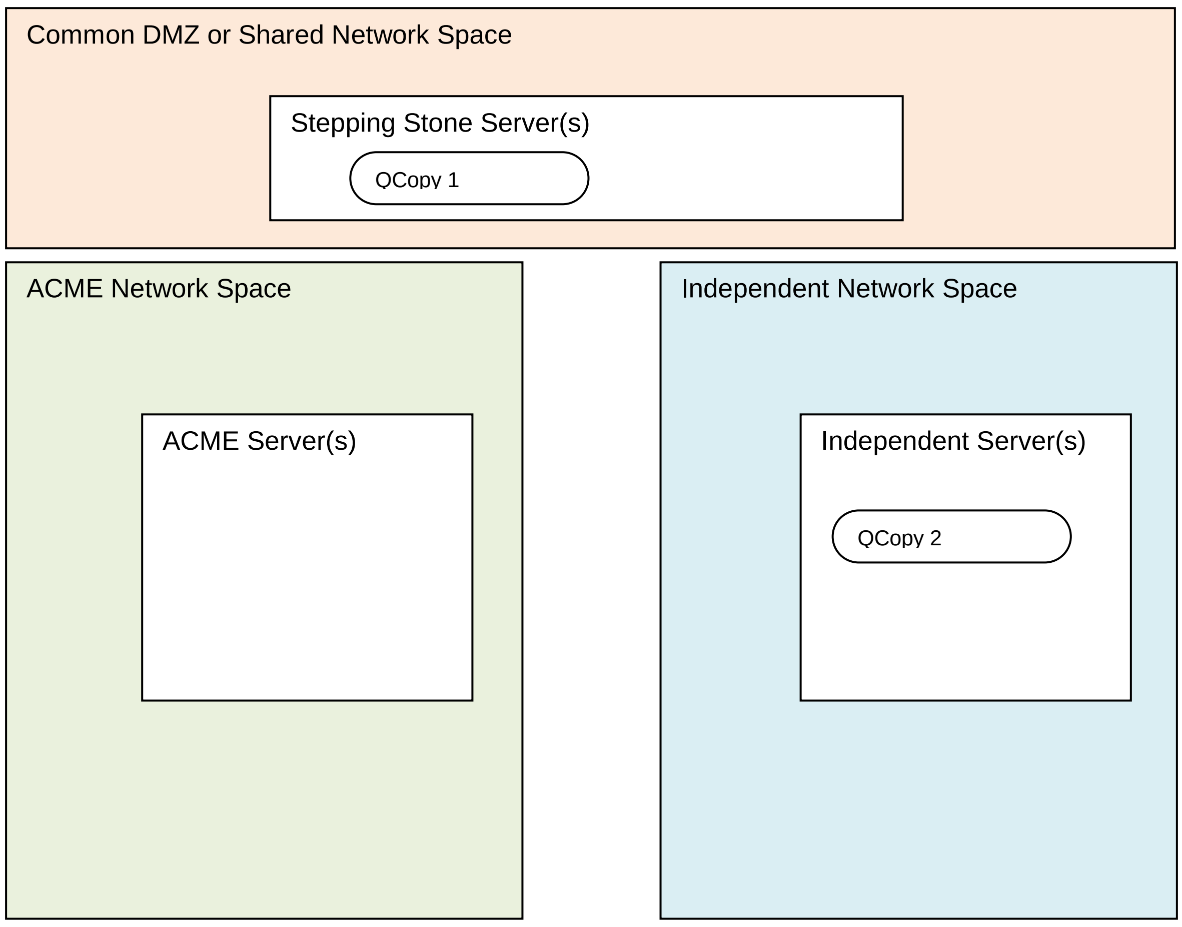
8.3.2 Dump

The dump command causes the processor to generate a Windows Core Minidump of the application. The application continues to execute so the command can be issued a number of times during an execution of the application. The contents of the dump can be investigated using the standard Windows debugging tools (e..g. windbg).

9. Common Usage Scenarios

9.1 De-Merger

In this scenario the “ACME Corporation” is spinning off a subsidiary to become the “Independent Corporation”, Network separation is already underway and Data Center facilities have already been created for the new entity. QCopy will be used to move Domino mail-files and application databases from the ACME infrastructure into the new Independent infrastructure.



In this scenario a scheduling and control application would sent transactions to the QCopy 1 instance instructing it to pull a replica of a particular database from one of the ACME servers onto the stepping stone server. Once the transaction was completed successfully then another transaction is sent to the QCopy 2 instance instructing it to pull a replica copy of the database from the stepping stone server onto one of the Independent servers. Typically once these steps were completed then other transactions would be triggered to, for instance, re-direct mail delivery to the Independent servers and switch client access to the instances on the Independent servers and delete the stepping stone and ACME server instances.

9.2 Reinstating a Test System

In this scenario a test server contains a directory called “Reference” which contains a number of sub-directories containing several databases that are all in the validated initial state for running a suite of regression tests. The tests are performed against the databases in a directory called “Test” which has the same subdirectory structure as the “Reference” directory. In this case QCopy is running on the test server and a single transactions (“Feeder Transaction”) is generated that specifies the source server as “Local” and the source as “Reference” and the target server as “Local” and the target as “Test” – pre-processing options are set to delete the target database before copying, other options are selected as appropriate.

9.3 Consolidation to a Global Data Center

In this scenario ACME Corporation is consolidating processing from a number of regional and local data centers into a single global data center. In this case instances of QCopy are deployed onto the new servers at the global site, probably with multiple instances running on each of the global servers. The instances of QCopy are fed with transactions to pull replica instances of the databases from the regional and local servers.

In this particular scenario the technical setup and implementation of QCopy is simple, here throughput optimisation is the key to success. As the overall project is about consolidation to a centralised site it is likely that there will be copies that have to be made over long (i.e. slow) network connections. Refer to the section on “Tuning Considerations” later in this guide for more information on this topic. The likely deployment pattern that would be created in this case would be that each of the global servers would be running a number of QCopy instances, each one would be running with different settings and would be dedicated to transfers from servers in a particular geographic region i.e. with particular network characteristics.

10. Tuning Considerations

The DXCommon kernel used in QCopy is the product of a research project that is investigating methods of autonomic optimisation in multi-threaded applications. There are many internal parameters that can be manipulated at the source level in order to effect changes in performance profile, consult the "DXCommon Application Developers Guide" for more information. The distributed version of the application only allows the manipulation of two parameters at runtime, the number of threads that will be used and the number of transactions that can be executed simultaneously (-M:nn and –X:nn parameters). The defaults for these settings are 10 threads (-M:10) and 1 transaction (-X:1).

10.1 Long Fat Pipes

The term "Long Fat Pipes" refers to network connections that have plenty of capacity but a high round-trip time, this type of connection is often encountered with intercontinental connections. Performance investigations into TCP/IP on "long Fat Pipes" show that applications typically have problems filling the pipe and therefore suffer from slow throughput as turnarounds on the connection lead to high idle times during data transfers across the connection. QCopy can be started with more threads than the default to help alleviate this slow throughput. Each thread uses a separate connection between the source and target server and therefore a separate TCP/IP state this results in a lower probability that the network connection will be in a completely idle state. For an individual connection you should experiment with the thread setting to determine the minimum number of threads that maximises the throughput over the connection.

10.2 Small Databases

When performing transfers that involve copying large numbers of very small databases then throughput may suffer due to the overhead of serial operations on the database compared to the time spent in data transfer. To alleviate this constraint increase the number of transactions that can be executed simultaneously, this may also result in needing to increase the number of threads that are being used.