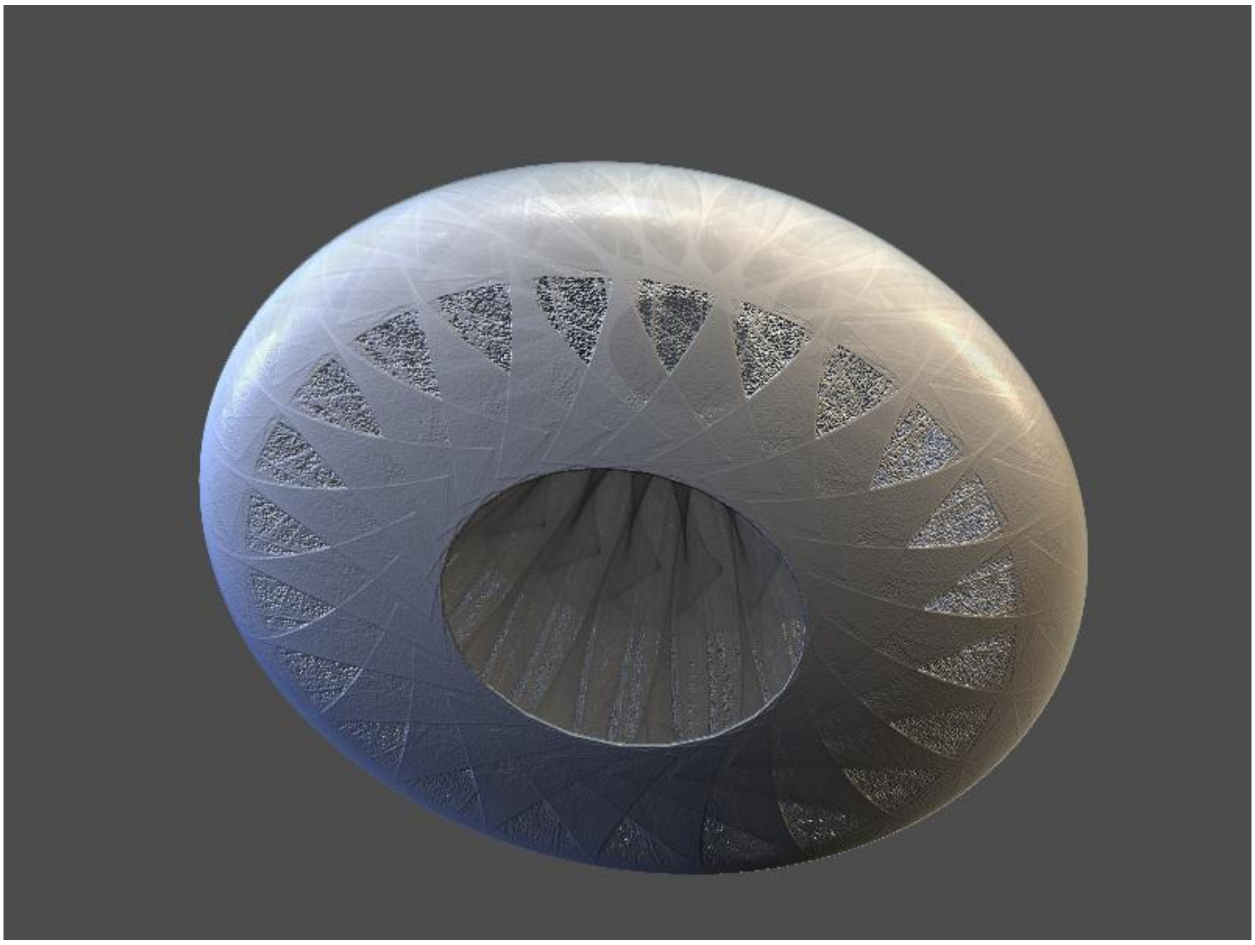
# DX Tools

# Using QACLMorph 1.0



**Author:** Ian Tree
**Owner:** HMNL b.*v.*
**Customer:** Public
**Status:** Final
**Date:** 24/01/2012 14:09
**Version:** 1.0

**Disposition:** Open Source

# Document History

## Document Usage

## Revision History

| | |
|---|---|
| Date of this revision: 23/03/2012 08:21 | Date of next revision *None* |

| Revision Number | Revision Date | Summary of Changes | Changes marked |
|---|---|---|---|
| 0.1 | 13/11/11 | Initial Base Version | No |
| 1.0 | 23/03/12 | QE Version | No |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

## Acknowlegements

Frontpiece Design was produced by the chaoscope application.

# Contents

# 1. Introduction to QACLMorph

QACLMorph is a tool for managing database ACLs in large volumes. The process is driven by an XML document that defines what a conformant ACL looks like for a set of databases it will then process each database to enforce conformance of the ACL. The utility was designed to operate against an XML document as these documents can be generated “on the fly”. The XML document defines a mixture of ACL patterns and entries that are

- Compulsory – must appear in the conforming ACL
- Forbidden – must NOT appear in the conforming ACL
- Permitted – if present in the ACL then they are conforming

QACLMorph is fast and resilient. Multiple databases can be processed in parallel allowing sustained high rates of ACL processing.

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

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

The ACL management engine at the heart of the QACLMorph application is mature and well tested component having been used in production environments for many years. It has been used in different configurations to manage the ACLs of over 100,000 databases.

# 2. QACLMorph Transactions

This section of the document identifies the different options that can be used on a QACLMorph transaction and how these options should be used.

## 2.1 Source Specification

These options are used to specify the database(s) that will be processed by the current transactions and the source of the ACL rules that will be applied to them.

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

### *2.1.1 Source Server*

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

### *2.1.2 Source Database*

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

### *2.1.3 ACL Specification*

A URL that identifies the XML document that contains the conforming ACL specification for the selected database(s). The specification can be a file name of a file on the local (where QACLMorph is running) or a HTTP URL.

### *2.1.4 Allow Recursion*

If the source database specification contains a directory then this setting determines if (Yes) the transaction will recurse into sub-directories or (No) not. Refer to the section on "Feeder Transactions" for more details.

## 2.2 Generic Transaction Options

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

### *2.2.1 Status*

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

### *2.2.2 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.

### 2.2.3 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”.

### 2.2.4 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.

## 2.3 Scheduling Options

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

### 2.3.1 First Run Time

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

### 2.3.2 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 intial 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. XML Document

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.

## 3.1 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.

## 3.2 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.

### *3.2.1 ACLRule Node*

The following attributes apply to an ACLRule Node.

#### 3.2.1.1 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 Set 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.

#### 3.2.1.2 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.

#### 3.2.1.3 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.

#### 3.2.1.4 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 .

### *3.2.2 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.

#### 3.2.2.1 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.

#### 3.2.2.2 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.

### *3.2.3 Role Node*

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

#### 3.2.3.1 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.

#### 3.2.3.2 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.

## 3.3 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.

### *3.3.1 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>
```

### 3.3.2 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 administrators from the ACL -->
<ACLRuleSet>
      <ForbiddenEntries>
            <ACLRule Pattern="*/ADMIN/ACME>
            </ACLRule>
      </ForbiddenEntries>
</ACLRuleSet>
```

### 3.3.3 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>
```

# 4. Feeder Transactions

## 4.1 General

“Feeder” transactions are single transactions that can be submitted to QACLMorph 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. QACLMorph will recognise such a transaction and process each database in the directory as if a single database had been specified in the transaction.

### *4.1.1 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 processing.

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 process 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 processing. Recursion into any sub-directories that are found within the specified scope can be prevented by setting the “Allow Recursion” option to “No”.

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.

### *4.1.2 Action and Error Processing Options*

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

### *4.1.3 Method of Operation*

When a Feeder transaction is executed then each database that matches the source selection is located and a new QACLMorph transaction is built, copying all of the options from the Feeder transaction. The XML document is only parsed once and a copy of the ACLRuleSet object is placed in each of the individual transactions. The new transactions are then posted for immediate processing.

# 5. Transaction Workflow

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

## 5.1 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.

## 5.2 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 QACLMorph 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.

## 5.3 In Progress Transaction

As soon as the QACLMorph 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.

## 5.4 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.

## 5.5 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.

## 5.6 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.

## 5.7 Delayed Transaction

If a transaction experienced a fault but was eligible to be retried then it is marked as “DELAYED” and moved to the delayed transaction queue. The QACLMorph 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.

# 6. Programmatic Interface

It is a relatively trivial task to interface the QACLMorph 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.

## 6.1 Creating New Transactions

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

- Form – “QATX”.
- 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 hosting the databases that are to be processed, set the field to “Local” to process databases on the current server.
- SourceDatabase – The name of the database to be processed.
- ACLPattern – Set this field to a file name or a HTTP URL specifying where the ACL rules XML document can be loaded from.

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.
- AllowRecursion – set this value to “Yes” to allow recursion into sub-directories found within scope or “No” to prevent this.

## 6.2 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.

# 7. Installing QACLMorph

## 7.1 Building the QACLMOrph Server Add-In Task

The QACLMorph 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 QACLMorph application.

### *7.1.1 Reference Environments*

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

**Windows:**

**Build Environment:**

Microsoft Visual Studio 2005

Version 8.0.50727.867  (vsvista.050727-8600)

Running on any supported windows workstation.

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

Notes API Version 8.5.

**Execution Environment:**

Windows Standard Server 2008 R2 (32 bit).

Domino Server 8.5.1 FP3.

**Note:** Execution environments from Domino 6.5.x through 8.5.x are regularly used.

**Linux:**

**Build Environment:**

Gcc Version: 4.1.2 for i386-redhat-linux.

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

Notes API Version 8.5

**Execution Environment:**

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

Domino Server 8.5.1 FP3.

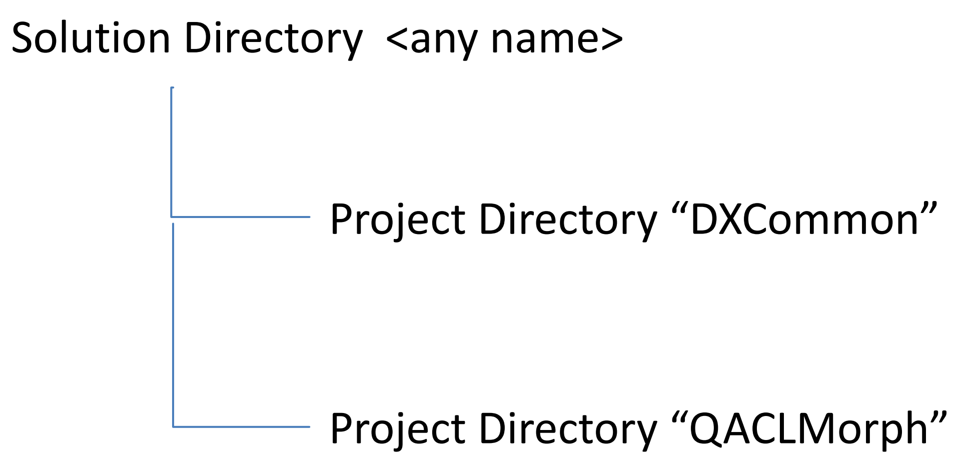
**Note:** Execution environments from Domino 7.0.x through 8.5.x are regularly used.

### *7.1.2 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.

### *7.1.3 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 "QACLMorph"

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.

### 7.1.4 Installing the DXCommon Kernel Sources

**Windows:**

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

As an example.

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

```
mklink /D DXCommon "c:\usr\include\DXcommon-3.12.0"
```

**Linux:**

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

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

As an example.

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

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

### 7.1.5 Installing the QACLMorph Sources

**Windows:**

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

ACLMorphRequest.h

AppCommandHandler.h

AppRunSettings.h

AppTransactionHandler.h

DbACLMorpher.h

QACLMorph.h

**Source Files**

ACLMorphRequest.cpp

AppCommandHandler.cpp

AppRunSettings.cpp

AppTransactionHandler.cpp

AbACLMorpher.cpp

QACLMorph.cpp

**Linux:**

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

### 7.1.6 Build Settings

**Windows:**

Add each source and header file that is used from the DXCommon kernel to the QACLMorph project. It is convenient to add these source and header files into subsets that can then be copied into other projects, in Visual Studio 2005 these collections are referred to as “filters”. To create a filter right-click on either the “Header Files” folder or the “Source Files” folder and select “Add” then “New Filter”. The following filters are convenient to use in the QACLMorph project “ACL”, “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\ACL**

DXCommon\ACL\DXACLRule.h

DXCommon\ACL\DXACLRuleSet.h

DXCommon\ACL\DXACLRuleSetParser.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\ACL**

DXCommon\ACL\DXACLRule.cpp

DXCommon\ACL\DXACLRuleSet.cpp

DXCommon\ACL\DXACLRuleSetParser.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.

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

**Notes:**

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

/Zp1 packing is a Notes API requirement as all Notes API structures are packed and not padded or member aligned.

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

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

**Linux:**

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

**make QACLMorph**

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

**make rebuild QACLMorph**

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

**make rebuild QACLMorph 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 QACLMorph 1.0.5 -- Build: 42 -- DXCommon 3.12.0
#
#  Optional targets:
#
#  rebuild     -  force a complete rebuild of the target
#
#  macros:
#
#  BV=DBG      -  Builds the DEBUG variant of the target
#

TARGET = QACLMorph

#  Define the primary source files

SOURCES = $(TARGET).cpp
SOURCES += AppRunSettings.cpp
SOURCES += AppCommandHandler.cpp
SOURCES += AppTransactionHandler.cpp
SOURCES += DbACLMorpher.cpp
SOURCES += ACLMorphRequest.cpp
HEADERS = $(TARGET).h
HEADERS += AppRunSettings.h
HEADERS += AppCommandHandler.h
HEADERS += AppTransactionHandler.h
HEADERS += DbACLMorpher.h
HEADERS += ACLMorphRequest.h

#  Define the ACL transformation modules to build

ACL_SOURCES = ../DXCommon/ACL/DXACLRule.cpp
ACL_SOURCES += ../DXCommon/ACL/DXACLRuleSet.cpp
ACL_SOURCES += ../DXCommon/ACL/DXACLRuleSetParser.cpp
ACL_HEADERS = ../DXCommon/ACL/DXACLRule.h
ACL_HEADERS += ../DXCommon/ACL/DXACLRuleSet.h
ACL_HEADERS += ../DXCommon/ACL/DXACLRuleSetParser.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_SOURCES += ../DXCommon/DXResource.cpp
CORE_SOURCES += ../DXCommon/DXResourceLoader.cpp
CORE_SOURCES += ../DXCommon/DXUCItem.cpp
CORE_SOURCES += ../DXCommon/DXUploadContext.cpp
CORE_HEADERS = ../DXCommon/APIPackages.h
CORE_HEADERS += ../DXCommon/DXException.h
CORE_HEADERS += ../DXCommon/ExecEnvironment.h
CORE_HEADERS += ../DXCommon/ElapsedTimer.h
CORE_HEADERS += ../DXCommon/RunSettings.h
CORE_HEADERS += ../DXCommon/DXGlobals.h
CORE_HEADERS += ../DXCommon/DXResource.h
CORE_HEADERS += ../DXCommon/DXResourceLoader.h
CORE_HEADERS += ../DXCommon/DXUCItem.h
CORE_HEADERS += ../DXCommon/DXUploadContext.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 += DbACLMorpher.o
OBJECTS += ACLMorphRequest.o
ACL_OBJECTS = DXACLRule.o
ACL_OBJECTS += DXACLRuleSet.o
ACL_OBJECTS += DXACLRuleSetParser.o
CORE_OBJECTS = APIPackages.o
CORE_OBJECTS += DXException.o
CORE_OBJECTS += ExecEnvironment.o
CORE_OBJECTS += ElapsedTimer.o
CORE_OBJECTS += RunSettings.o
CORE_OBJECTS += DXResource.o
CORE_OBJECTS += DXResourceLoader.o
CORE_OBJECTS += DXUCItem.o
CORE_OBJECTS += DXUploadContext.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 qaclmorph
INCDIR = $(LOTUS)/notesapi/include
LIBS = -lnotes -lm -lnsl -lpthread -lc -lresolv -ldl -lcurl
DEFINES = -DUNIX -DLINUX -DHANDLE_IS_32BITS

#  Rules to build DEBUG or release targets

$(TARGET): $(OBJECTS) $(ACL_OBJECTS) $(CORE_OBJECTS) $(MTCORE_OBJECTS) $(EXPLORER_OBJECTS)
	$(CC) $(LINKOPTS) $(OBJECTS) $(ACL_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
$(ACL_OBJECTS): $(ACL_SOURCES) $(ACL_HEADERS)
ifeq ($(BV), DBG)
	$(CC) $(CCOPTS) $(DEFINES) -D_DEBUG -I$(INCDIR) $(ACL_SOURCES)
else
	$(CC) $(CCOPTS) $(DEFINES) -I$(INCDIR) $(ACL_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 qaclmorph
```

**Notes:**

There is currently little difference between the release and debug builds of the QACLMorph 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 “qaclmorph”, this is a Domino convention.

### 7.1.7 *Building and Deploying the Application*

**Windows:**

Select “Build” and then “Build QACLMorph”.

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

**Linux:**

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

**make QACLMorph**

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

**make rebuild QACLMorph**

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

**make rebuild QACLMorph 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 (qaclmorph) 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 qaclmorph

chgrp notes qaclmorph

chmod 2555 qaclmorph

## 7.2 Installing the QACLMorph 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.

### 7.2.1 *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 QACLMorph Control Database, use the following URL:**

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

# 8. Starting QACLMorph on the Server

The QACLMorph 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 QACLMorph <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).

### *8.1.1 Command Line Options*

The following command line options are available with QACLMorph.

| 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 trace, 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. |
| **-X:nn** | Sets the number of transactions that can execute in parallel. This defaults to 1. Therefore, by default, QACLMorph transactions will execute serially. |

# 9. QACLMorph Tell Commands

## 9.1 QACLMorph Message Queues

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

QACLMorph 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 QACLMorph is started it locates the lowest number available to use for the message queue name in the form “QACLMorph”<suffix> where <suffix> is a digit that starts at 1. Therefore the first, or only, instance of QACLMorph can be addressed in the form “Tell QACLMorph1 <command>”.

## 9.2 Commands

### *9.2.1 Quit*

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

### *9.2.2 Stop [now]*

The stop command is used to shut down QACLMorph 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 QACLMorph to fail any transactions that are currently running and then shut down in an orderly manner.

### *9.2.3 Abort*

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

### *9.2.4 Suspend*

The suspend command tells QACLMorph 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.

### *9.2.5 Resume*

The resume command is the antithesis 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.

### *9.2.6 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.

### 9.2.7 Loud

The loud command is an alias for the verbose command.

### 9.2.8 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.

### 9.2.9 Quiet

The quiet command is an alias for the terse command.

### 9.2.10 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.

### 9.2.11 Noecho

The noecho command is an alias for “echo off”.

### 9.2.12 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.

### 9.2.13 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.

### 9.2.14 Refresh

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

### 9.2.15 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:**

```
07/12/2010 08:51:50 CET: DXR0907I: Command received: status. [500]
07/12/2010 08:51:50 CET: QAM0201I: 1 transaction have been dispatched, 1 completed, 0 are
running, max concurrency is 1. [500]
07/12/2010 08:51:50 CET: QAM0208I: Transactions marked Completed: 0, Error: 1, Retried: 0,
Delayed: 0. [500]
```

### *9.2.16 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.

### *9.2.17 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.

### *9.2.18 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.

## 9.3 Debugging Commands

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

### *9.3.1 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.

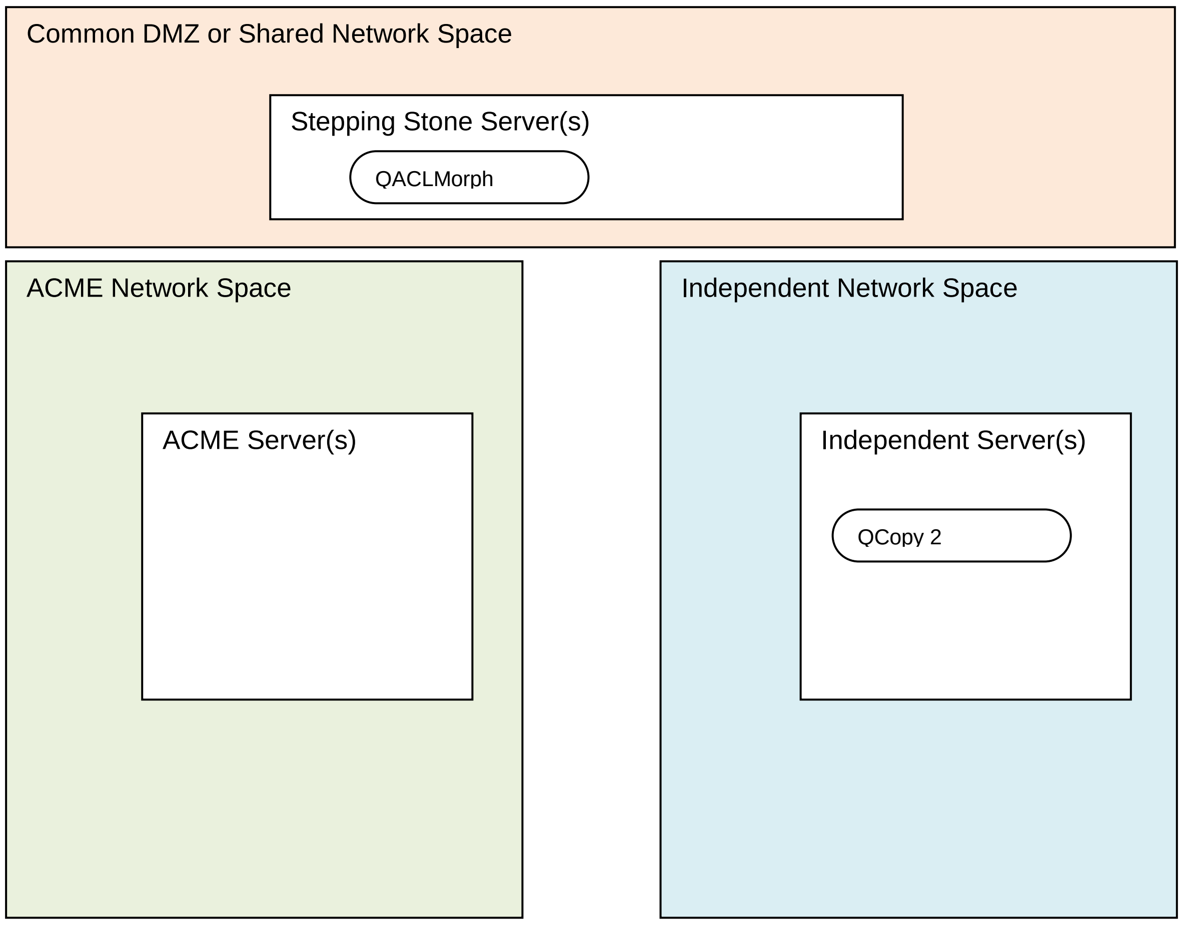
### *9.3.2 Dump*

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).

# 10. Common Usage Scenarios

## 10.1 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. Before the databases will be copied from the Stepping Stone server(s) to the ACL of all the databases will have their ACLs adjusted so that ACME related entries are removed and the Independent ACL standards will be applied. Old administration groups and servers will be removed and replaced with new groups and servers.


Common DMZ or Shared Network Space
Stepping Stone Server(s)
QACLMorph
ACME Network Space
ACME Server(s)
Independent Network Space
Independent Server(s)
QCopy 2