# Calling MQSeries from Software AG's Natural
## Version 1.0

6th December, 2001

Michael Fabianski
IBM UK Ltd
Warwick

michael_fabianski@uk.ibm.com

Ed Fletcher
IBM UK Ltd
Hursley

ed_fletcher@uk.ibm.com

Martin Howson
IBM UK Ltd
Warwick

howsonm@uk.ibm.com

**Take Note!**

Before using this report be sure to read the general information under "Notices".

**First Edition, November 2001**

This edition applies to Version 1.0 of *Calling MQSeries from Software AG's Natural* and to all subsequent releases and modifications unless otherwise indicated in new editions.

# Table of Contents

# Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates.

Any reference to an IBM licensed program or other IBM product in this publication is not intended to state or imply that only IBM's program or other product may be used. Any functionally equivalent program that does not infringe any of the intellectual property rights may be used instead of the IBM product.

Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, New York 10594, USA.

The information contained in this document has not be submitted to any formal IBM test and is distributed AS-IS. The use of the information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item has been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

## Trademarks and service marks

The following terms, used in this publication, are trademarks of the IBM Corporation in the United States or other countries or both:

- IBM
- MQSeries
- MQSeries Integrator
- MQSeries Workflow

The following terms are trademarks of other companies:

| | |
|---|---|
| Windows NT, Windows 2000 | Microsoft Corporation |
| Natural | Software AG |
| Solaris | Sun Microsystems |

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

# Preface

This SupportPac looks at the interaction between MQSeries and Software AG’s Natural. It shows how calls to MQSeries can be made from within the Natural environment and therefore how it can be linked to the whole enterprise.

It consists of a number of source files for Natural, an example showing how MQSeries calls can be made from the Natural environment on Windows NT/2000 and documents the differences on z/OS, UNIX systems.

# Bibliography

The expected audience for this document is Natural trained professionals needing to link with MQSeries. It is expected that the reader is familiar with the concepts and activities involved in setting up and running Natural and MQSeries.

The following sources of information may be useful:

- Natural product manuals, especially:
  - *Natural Programming Guide*, NAT226-020
- MQSeries product manuals, especially
  - *MQSeries Application Programming Guide*, SC33-0807-12
  - *MQSeries Application Programming Reference*, SC33-1673-08

All of the IBM books are available online. The MQSeries books are at the following URL:

http://www.ibm.com/software/mqseries/library/manualsa/

Education covering MQSeries is available from IBM. Education covering Natural is available from Software AG.

# Acknowledgements

A number of people have contributed to the development of this SupportPac. The authors would like to thank Lee Schofield and Mohammed Ajab of IBM.

# Summary of Amendments

| **Date** | **Changes** |
|---|---|
| 06 December 2001 | Initial release |

# Chapter 2. Introduction

## Overview

This SupportPac originates from work done in several customers in 2000 and 2001 who had Natural applications running on both mainframes and on Sun Solaris, who then wished to call MQSeries from those applications.

IBM along with the customers wrote some Natural programs, and created some Natural Local Data Areas, to allow MQ Series to be called. This SupportPac contains some of the sample code that was created to make the various MQSeries calls from Natural, it also contains some of the Local data areas that are required when calling MQSeries with the Connect, Open, Put, Get, Close and Disconnect verbs. For the other MQSeries commands, the above can be used as templates.

The Natural code to make the calls to MQSeries remains the same regardless of the platform on which the Natural code is running, the installation and linking of MQSeries does however differ greatly on the different platforms.

On the mainframe it is a simple procedure to link the MQSeries modules into the Natural nucleus. The statement CALL 'MQOPEN' ... for example, is able to call MQSeries directly and pass parameters to it. The way that the MQSeries load libraries are linked into Natural to achieve this is documented later in this SupportPac. In customer trials, links from both batch Natural and Natural running under CICS were successfully achieved.

On the Windows NT/2000 and UNIX operating system however, it is slightly more complicated to call MQSeries from Natural. The MQSeries calls are provided in a shared library but Natural cannot call a shared library and locate the correct entry point from a 'CALL' statement. A number of 'C' functions need to be written to call the MQSeries shared library with the correct parameters. These 'C' functions can then be called from a Natural program. For example, a number of modules called MQBACK, MQBEGIN, MQOPEN, MQCLOSE, MQCONN, MQPUT, MQGET, MQDISC and MQCMIT could be created to be called from Natural. Each of these modules takes the parameters passed to them and calls the appropriate MQSeries function with those parameters. These C functions are provided within this SupportPac.

## Installation

The SupportPac is supplied in the zip file, md07.zip. This should be uncompressed to a directory of choice and will produce the following files:

| | |
|---|---|
| makefile | Make file to create DLLs for Natural |
| mqnamq.c | C program that forwards calls to MQxxxx functions to corresponding NAMQxxxx functions |
| mqnamq.def | def file for the creation of mqnamq.dll |
| mqnamq.dll | DLL that forwards calls to MQxxxx functions to corresponding NAMQxxxx functions |
| mqnamq.exp | exp file corresponding to mqnamq.dll |
| mqnamq.lib | lib file corresponding to mqnamq.dll |
| mqnamq.obj | Object file corresponding to mqnamq.dll |

| | |
|---|---|
| mqwrapback.c<br>mqwrapback.obj | C program that performs MQBACK call<br>Object file for the MQBACK call |
| mqwrapbegin.c<br>mqwrapbegin.obj | C program that performs MQBEGIN call<br>Object file for the MQBEGIN call |
| mqwrapclose.c<br>mqwrapclose.obj | C program that performs MQCLOSE call<br>Object file for the MQCLOSE call |
| mqwrapcommit.c<br>mqwrapcommit.obj | C program that performs MQCMIT call<br>Object file for the MQCMIT call |
| mqwrapconnect.c<br>mqwrapconnect.obj | C program that performs MQCONN call<br>Object file for the MQCONN call |
| mqwrapdisconnect.c<br>mqwrapdisconnect.obj | C program that performs MQDISC call<br>Object file for the MQDISC call |
| mqwrapget.c<br>mqwrapget.obj | C program that performs MQGET call<br>Object file for the MQGET call |
| mqwrapopen.c<br>mqwrapopen.obj | C program that performs MQOPEN call<br>Object file for the MQOPEN call |
| mqwrapput.c<br>mqwrapput.obj | C program that performs MQPUT call<br>Object file for the MQPUT call |
| mqwrapput1.c<br>mqwrapput1.obj | C program that performs MQPUT1 call<br>Object file for the MQPUT1 call |
| namq.def | def file for the creation of namq.dll |
| namq.dll | DLL that performs calls to MQSeries functions |
| namq.exp | exp file corresponding to namq.dll |
| namq.h | header file for C programs |
| namq.lib | lib file corresponding to namq.dll |
| namqdebug.c<br>namqdebug.obj | C program containing tracing functions for sample code<br>Object file for the tracing functions for sample code |

# Chapter 3. The MQSeries and Natural Environment

## MQSeries – Messaging

MQSeries messaging is now installed in over 7000 customers. Its success has risen from its ability to provide a connectivity solution that bridges the information gap between transaction-based mainframe systems and client-server applications running on heterogeneous collections of UNIX and PC platforms. MQSeries is supported on over 35 different platforms.

MQSeries pervasiveness has arisen from its ability to address the ever-increasing quantity and types of information flowing within and between organisations in a uniform, highly robust, manner, giving the quality of service essential to support the business processes. One US government agency, for example, is processing in excess of 100 million messages a day. A bank reports averaging 144 million messages a day with peaks of 250 million in their organisation. Most users of MQSeries found the 4 MB maximum message size to be more than adequate, however this has been increased recently to 100 MB on many platforms in response to customer requests to handle large documents such as CAD drawings in a single transaction. MQSeries has over 250 partners integrating MQSeries into their products or providing value-add tools. The new product announcements are intended to protect their investment and provide new opportunities.

MQSeries programming interfaces remove the need to program to low-level network protocols, or remote procedure calls. Applications exchange information without hard coding prior knowledge of each other, without prior knowledge of which platform, or where on the LAN or WAN they execute. Not only has this been very attractive for bespoke development within organisations, it has been the foundation for ISV partners to develop MQSeries “ready” applications.

IBM are introducing new, high level programming interfaces, which aim to increase development productivity, enabling MQSeries applications to maximise flexibility and responsiveness to change.

***IBM MQSeries***
***Redefining the way work is done***

With Version 5, MQSeries has been enhanced with publish /subscribe functionality to provide programming styles suited to one-to-many and many-to-many messaging models. A key differentiator is MQSeries ability to combine assured delivery of messages between two or more applications, with the fire and forget capability to publish non-transactional messages to an unknown number of subscribing applications. MQSeries developers can benefit from using the same API to work with all messaging models, minimizing learning curves. New customers benefit from a single-supplier solution. IBM are extending MQSeries to include both topic and content based publish /subscribe solutions.

This SupportPac is designed to assist customers who use Software AG’s Natural programming environment to use MQSeries, and therefore opens up the world of MQSeries Integrator and MQSeries Workflow.

## Software AG’s Natural

Natural is Software AG’s integral application development environment, which supports procedural and non-event-driven (Windows only) programming techniques. It empowers developer to create complex component-based business applications.

In order to develop and deploy client-server applications a powerful development environment and a high performance server machine are required. Natural is well placed to do this with its platform independent language Natural. Applications can be developed in one environment and deployed to all the target systems supported by Natural.

Natural is easy to learn, and applications written in Natural are easy to maintain. Natural has an open architecture, which allows it to provide everything expected of a strong two-tier development environment. Thus, the add-on products available in the Natural world mean that most customer situations are covered.

New technologies such as the Web or Component development are integrated into Natural. These technologies are implemented by extending the scope of the proven Natural technology.

Further information on Software AG's Natural can be found at the Software AG Web site at:

http://www.softwareag.com/natural/product/strategy.htm

# Chapter 4. Connecting Distributed MQSeries to Natural

## Overview

On the z/OS environment it is possible to call the MQSeries modules directly from Natural, as the MQ stub, which is a re-entrant module, is linked into the Natural nucleus, and the entry points into the module can be defined. The Natural architecture on the UNIX/Windows environments is quite different, and it is necessary to define some intermediate ‘C’ functions to make the MQ calls, as Natural has to have external calls mapped onto specific ‘C’ objects, rather than linking in to a library of objects.

This example uses a number of ‘C’ functions, one for each of the MQSeries commands rather than a generic bridge for all the MQSeries functions. These individual ‘C’ functions take parameters from Natural, and pass them on unchanged to the correct MQSeries object. The design was done in this way to allow Natural to call MQSeries as if it was calling MQSeries directly. So in Natural the MQPUT, MQGET etc would be coded as normal, providing Natural with the ability to use the full functionality of MQSeries, and to give Natural full control of MQSeries.

The ‘C’ routines have been made as simple as possible, so that little or no maintenance of these routines would be required. As soon as application logic is placed into these ‘C’ routines, the likelihood that the routines would need to be maintained at some point in the future increases. These routines will just marshal parameters back and forth; any MQSeries errors that occur will be passed back to Natural and handled there.

In addition, having a one for one mapping of the ‘C’ functions to the MQSeries calls, means that whenever new functionality is introduced into MQSeries, Natural will be able to use it without changes to the existing ‘C’ routines.

## Examples of the Natural statements used to call MQ Series

The following calls are the ones that will need to be coded in Natural to call MQSeries for the Connect, Open, Close, Put and Get commands:

```
CALL 'MQCONN' #QMGRNAME #HCONN #COMPCODE #REASON

CALL 'MQOPEN' #HCONN #OBJDESC #OPTIONS #REQUEST #HOBJ  #COMPCODE
#REASON

CALL 'MQCLOSE' #HCONN #REQUEST-HOBJ #OPTIONS  #COMPCODE #REASON

CALL 'MQGET' #HCONN #REQUEST-HOBJ #MSGDESC #GETMSGOPTS

                #BUFFERLENGTH #BUFFER (1:#BUFFERLENGTH)

                #DATALENGTH #COMPCODE #REASON

CALL 'MQPUT' #HCONN #REPLY-HOBJ #MSGDESC #PUTMSGOPTS  #BUFFERLENGTH

                #BUFFER(1:#BUFFERLENGTH)  #COMPCODE #REASON
```

The variable in the calls were defined as follows:

```
#QMGRNAME  (A48)
#HCONN  (I4)
#COMPCODE  (I4)
#REASON  (I4)
#OBJDESC  (A168)
#OPTIONS  (I4)
#HOBJ  (I4)
#MSGDESC  (A72)
#GETMSGOPTS  (A128)
#PUTMSGOPTS  (A128)
#DATALENGTH  (I4)
#BUFFERLENGTH  (I4)
#BUFFER  (A1/1:2000)
#REQUEST-OPEN  (L) INIT <FALSE>
#REPLY-OPEN  (L) INIT <FALSE>
#REQUEST-HOBJ  (I4) 0900 1 #REPLY-HOBJ  (I4)
```

The variable PUTMSGOPTS has been redefined in Natural with the following structure:

```
      1 MQPMO                          A128
R   1 MQPMO
        2 MQPMO-STRUCID                      A4
        2 MQPMO-VERSION                      I4
        2 MQPMO-OPTIONS                      I4
        2 MQPMO-TIMEOUT                      I4
        2 MQPMO-CONTEXT                      I4
        2 MQPMO-KNOWNDESTCOUNT         I4
        2 MQPMO-UNKNOWNDESTCOUNT             I4
        2 MQPMO-INVALIDDESTCOUNT       I4
        2 MQPMO-RESOLVEDQNAME          A48
        2 MQPMO-RESOLVEDQMGRNAME             A48
```

Structures are similarly defined for all the other MQ parameters such as MQMD etc.

## Linking Natural with MQSeries on UNIX

Within the Natural environment, to call external functions user exits are needed to make these functions available. Although the Natural documentation says that the user exits can either be placed in a shared library and linked dynamically, it was found that this didn't seem to work, so 'C' objects were created and linked statically with the Natural nucleus.

Natural calls external functions via the use of a jump table, which provides a mapping on the object name used within Natural and the real called object. This jump table is a 'C' module and is included in Appendix 2. It can be seen in the jumptable entry in Appendix 2 that the MQSeries call MQCONN maps onto a function called NAMQCONN, which is a 'C' routine that gets parameters from Natural and calls the MQCONN routine via the library –lmqm.

This jumptable is linked into the Natural Nucleus; see the makefile in Appendix 1. Whenever a 'CALL' statement is then issued in the Natural program, the module defined in the jumptab will be called. In the example shown here the

```
CALL 'MQCONN' #QMGRNAME #HCONN #COMPCODE #REASON
```

Statement will go to the jump table, see that MQCONN maps onto NAMQCONN and will call that module passing on the parameters listed after the call. The module NAMQCONN (See Appendix 3) then takes the parameters sent to it and calls the MQSeries function.

# Chapter 5. An MQSeries and Natural Example on Windows 2000

The following objects will be loaded into a Natural user library named MQSERIES:

| | |
|---|---|
| INIT-GMO.NSC | Natural sample Copycode which initialises the MQSeries MQGMO structure |
| INIT-MD.NSC | Natural sample Copycode which initialises the MQSeries MQMD structure |
| INIT-PMO.NSC | Natural sample Copycode which initialises the MQSeries MQPMO structure |
| MQ-CONN.NS3 | Natural sample Dialog which prompts for the name of an MQSeries Queue manager |
| MQ-OPEN.NS3 | Natural sample Dialog which prompts for the name of an MQSeries Queue |
| MQ-WRITE.NS3 | Natural sample Dialog which prompts for text to be placed on the currently open MQSeries queue |
| MQCCV.NSL | Natural sample Local Data Area which defines MQSeries return codes |
| MQCOV.NSL | Natural sample Local Data Area which defines MQCLOSE option values |
| MQDIALOG.NS3 | Natural sample Dialog which provide the capability to perform MQSeries calls |
| MQGMO.NSL | Natural sample Local Data Area which defines the MQSeries MQGMO structure |
| MQGMOV.NSL | Natural sample Local Data Area which defines initial values for the MQSeries MQGMO structure |
| MQMD.NSL | Natural sample Local Data Area which defines the MQSeries MQMD structure |
| MQMDV.NSL | Natural sample Local Data Area which defines initial values for the MQSeries MQMD structure |
| MQOD.NSL | Natural sample Local Data Area which defines the MQSeries MQOD structure |
| MQOOV.NSL | Natural sample Local Data Area which defines MQOPEN option values |
| MQPMO.NSL | Natural sample Local Data Area which defines the MQSeries MQPMO structure |
| MQPMOV.NSL | Natural sample Local Data Area which defines initial values for the MQSeries MQPMO structure |
| NAMQTEST.NSP | Natural sample program that invokes MQSeries API calls |

# Installation instructions

1. Copy NAMQ.DLL and MQNAMQ.DLL to your Natural bin directory

2. Set the environment variable NATUSER

   Start the Control Panel, open System Properties and enter the variable NATUSER into the environment settings.

   Variable: natuser
   Value: mqnamq

   NOTE: if you want to specify other additional libraries in your NATUSER variable, you have to separate the names with a semicolon (;),

   For example: "Value: mqnamq;userlib1;userlib2;userlib3".

3. Start Natural

4. Invoke the SYSTRANS utility:
   Enter the command: LOGON SYSTRANS
   Enter the command: MENU
   Select 'Load TRANSFER objects' and click on 'OK'
   Enter the name of the transfer file - use 'Browse' to locate the file (Natural.dat) - and click on 'OK'
   Click on 'Load'

5. Exit from the SYSTRANS utility

6. Select the MQSERIES user library

7. Right click on the MQSERIES user library and select 'Cat All'

8. Execute the sample dialog:
   Expand the MQSERIES User library to show the available dialogs
   Select and run the MQDIALOG dialog
   Use the 'File' menu items to perform MQSeries calls

# Chapter 6. Configuration Steps Required for z/OS

This document outlines what was necessary to allow Natural to communicate with MQSeries both online and in batch.

1. Create a sample Natural program that calls MQSeries.

   The following program, for example, was called MQMTEST and was stowed in library DBAAPPL..

**Note:** the queue manager name, which is ENGQ in our example below may differ between sites, this can be found from the MQ-Series master address space.

This simple program connects to the MQSeries Queue Manager, and disconnects. Providing that the return codes (#COMPCODE and #REASON) are both zero, the connection to MQSeries has worked.

```
DEFINE DATA LOCAL
1 #QMNAME   (A48) INIT <'ENGQ'>
1 #HCONN    (I4)
1 #COMPCODE (I4)
1 #REASON   (I4)
END-DEFINE
  WRITE *PROGRAM
  CALL 'MQCONN' #QMNAME #HCONN #COMPCODE #REASON
  WRITE 'after MQCONN call' #COMPCODE #REASON
  CALL 'MQDISC' #HCONN #COMPCODE #REASON
  WRITE 'after MQDISC call' #COMPCODE #REASON
END
```

2. For Batch

   2.1. Edit the Natural Batch parameter module source and insert the following. Please ensure that you retain the CSTATIC entries that may be already listed within your source.

```
ATTN=ON,               MQSERIES ATTN KEY    -
NAFSIZE=1,             MQSERIES NAF BUFFER  -
CSTATIC=(CMMSG,        MQSERIES             -
CTMOD,                 MQSERIES             -
CXLU62,                MQSERIES             -
MQBACK,                MQSERIES             -
MQCMIT,                MQSERIES             -
MQCLOSE,               MQSERIES             -
MQCONN,                MQSERIES             -
MQDISC,                MQSERIES             -
MQGET,                 MQSERIES             -
MQINQ,                 MQSERIES             -
MQOPEN,                MQSERIES             -
MQPUT,                 MQSERIES             -
MQPUT1,                MQSERIES             -
MQSET),                MQSERIES             -
```

2.2. Edit the JCL that compiles and link edits the batch parameter module source, ensure that it includes both of the lines below which contain the MQMLIB string.

```
//PRM312BA  JOB EXBAAM00,'DBA',MSGCLASS=K,TIME=1440,REGION=0M
//*MAIN     CLASS=LIVE
//PROCS     JCLLIB ORDER=(ISDPR.PROCLIB)
//*-------------------------------------------------------------
//ASMPRMC   EXEC PGM=ASMA90,
//          PARM='OBJECT'
//SYSIN     DD   DSN=SAG.NXT.TST.SRCE(PRM312BA),
//          DISP=SHR
//SYSLIB    DD   DSN=SAG.NAT312.SRCE,
//          DCB=BLKSIZE=30000,DISP=SHR
//          DD   DSN=SYS1.MACLIB,DISP=SHR
//SYSUT1    DD   UNIT=SYSDA,SPACE=(1700,(600,100))
//SYSUT2    DD   UNIT=SYSDA,SPACE=(1700,(600,100))
//SYSUT3    DD   UNIT=SYSDA,SPACE=(1700,(600,100))
//SYSTERM   DD   SYSOUT=*,DCB=BLKSIZE=1809
//SYSPRINT  DD   SYSOUT=*,DCB=BLKSIZE=1809
//SYSPUNCH  DD   DUMMY
//SYSLIN    DD   DSN=&&PD,UNIT=SYSDA,
//          DISP=(NEW,PASS,DELETE),SPACE=(1700,(1000,300),RLSE)
//*
//LKDPRMC   EXEC PGM=IEWL,
//          PARM='RENT,REUS,XREF,LET,LIST,NCAL,SIZE=(512K,128K)',
//          COND=(4,LT,ASMPRMC)
//SYSUT1    DD   UNIT=SYSDA,SPACE=(1700,(500,100))
//SYSPRINT  DD   SYSOUT=*,DCB=(RECFM=FB,LRECL=121,BLKSIZE=1210)
//SYSLMOD   DD   DSN=SAG.NXT.TST.LOADLIB,
//          DISP=SHR
//MQMLIB    DD   DSN=SYS1.MQM.SCSQLOAD,DISP=SHR
//SYSLIN    DD   DSN=&&PD,DISP=(OLD,DELETE)
//          DD   *
 INCLUDE MQMLIB(CSQBSTUB)   MQSERIES BATCH STUB
 NAME PRM312BA(R)
/*
```

2.3. Submit the modified job and ensure rc00.

2.4. Test the Natural program created in action 1 above via a Natural batch job.

**Note:** two additional libraries need to be appended to the STEPLIB; these are SYS1.MQM.SCSQAUTH and SYS1.MQM.SCSQANLE. Below is a sample batch job that calls a PROC, in our example variables LIB1 and LIB2 are appended to our STEPLIB concatenation.

```
//NATBATCH  JOB EXBAAM00,'DBA',MSGCLASS=K,TIME=1440

//*MAIN     CLASS=LIVE

//PROCS     JCLLIB ORDER=(ISDPR.ISD.PROCLIB)

//*

//BATCH       EXEC NATBATCH,ENV=TST,SYS=NXT,UDB=NXT,

//          LIB1=SYS1.MQM.SCSQAUTH,

//          LIB2=SYS1.MQM.SCSQANLE

//SYSIN      DD   *

LOGON DBAAPPL

MQMTEST

FIN

/*
```

The expected output is as follows:

```
Logon accepted to library DBAAPPL.

NEXT LOGON DBAAPPL

Logon accepted to library DBAAPPL.

NEXT MQMTEST

Page      1

MQMTEST

after MQCONN call          0          0

after MQDISC call          0          0

NEXT FIN

NAT9995 Natural SESSION TERMINATED NORMALLY
```

3. For CICS

   3.1. Repeat action 2.1 on the Natural CICS parameter module source, e.g., PRM312CI

   3.2. Repeat action 2.2 on the JCL that compiles and link edits the Natural CICS parameter module, e.g., PRM312CI

   **Note:** The MQ-SERIES stub is ***CSQCSTUB*** not CSQBSTUB

   3.3. Submit the JCL and ensure rc00.

   3.4. Newcopy the Natural CICS parameter module.

   3.5. Logon to the CICS region and test the sample program that was created in action 1 above.

# Appendix 1 The Natural Makefile

```
##################################################### (c) Software AG 1998
# File            : $NATDIR/$NATVERS/samples/sysexuex/Makefile
# Creation date   : 14:33:37 29.10.100
# Machine         : sun4_sol
# Natural version : 311
##########################################################################
#
#  Generate executable Natural program from natraw.o, CM library,
#  userexits and database drivers.
#
#  To install Natural with the necessary drivers appropiate to your
#  Database sytstem(s) you have to run this Makefile with some of
#  the following flags.
#
#     make natural {<flags>}
#
#  Please refer to the section about the specific database system for
#  further informations about the flags.
#
#  Following Environment Variables are needed for successful linking
#  of Natural
#
#    NATDIR     : Natural base directory
#    NATVERS    : Natural version
#
#------------------------------------------------------------------------
#
#  Using ADABAS :
#
#  if you want to use ADABAS with the Natural you are generating, you have to
#  specify the mode how ADABAS should be linked by using one of the following
#  command line flags
#
#    ada=dyn       use dynamic binding of ADABAS (version 1.2 and higher)
#    ada=stat      use static binding of ADABAS (prior to version 1.2)
#    ada=cscidyn   use dynamic CSCI interface also for ADABAS access
#    ada=cscistat  use static CSCI interface also for ADABAS access
#
#  The following Environment variables must be set :
#
#    ADADIR     : ADABAS base directory
#    ADAVERS    : ADABAS version
#
#  Additional Environment variables for CSCI :
#
#    NETDIR  : NET-WORK base directory
#    NETVERS : NET-WORK version
#
#------------------------------------------------------------------------
#
#  Using ENTIRE database :
#
#  if you want to use ENTIRE you have to specify following command line
#  flag
#
#    ent=yes
#
#  The following Environment variables must be set :
#
#    AERDIR     : ENTIRE base directory
#    AERVERS    : ENTIRE version
#
```

```
#-------------------------------------------------------------------------
#
#  Using SQL databases with OSQ :
#
#  to link the SQL interface to the Natural system, please
#  specify following flag at the command line. This option is
#  usable only if the SQL interface libraries are accessible
#  at the $NATDIR/$NATVERS/bin/build directory.
#
#     sql=yes
#
#  The following Environment variables must be set :
#
#    OSQDIR     : ENTIRE ACCESS base directory
#    OSQVERS    : ENTIRE ACCESS version
#
#-------------------------------------------------------------------------
#
#  Using SQL databases with OSX :
#
#  to link the SQL interface (OSX version) to the Natural system,
#  please specify following flag at the command line. This option
#  is usable only if the SQL interface libraries are accessible
#  at the $NATDIR/$NATVERS/bin/build directory.
#
#     osx=yes
#
#  The following Environment variables must be set :
#
#    OSXDIR     : OSX base directory
#    OSXVERS    : OSX version
#
#-------------------------------------------------------------------------
#
#  Using ADABAS SQL server :
#
#  to link the ADABAS SQL server interface to the Natural system, please
#  specify following flag at the command line.
#
#     esq=yes
#
#  The following Environment variables must be set :
#
#    ESQDIR     : ADABAS SQL server base directory
#    ESQVERS    : ADABAS SQL server version
#
#    NETDIR  : NET-WORK base directory
#    NETVERS : NET-WORK version
#
#-------------------------------------------------------------------------
#
#  Using TP monitor :
#
#  to link the TP monitor interface to the Natural system, please
#  specify following flag at the command line. This option is
#  usable only if the TP monitor interface libraries are accessible
#  at the $NATDIR/$NATVERS/bin/build directory.
#
#     tp=yes
#
#-------------------------------------------------------------------------
#
#  Using SYNCSORT :
#
#  to use the external SYNCSORT product, please specify following
#  flag at the command line.
#  The default library search path will be used for accessing the library.
#
```

```
#     sync=yes
#
#------------------------------------------------------------------------
#
#  Using BROKER Stubs :
#
#  To use Broker stubs a new Natural nucleus has be linked. This nucleus
#  will support multithreading. In order to use the Broker stubs please
#  specify the following on command line (in addition to other settings)
#
#     exx=yes
#
#------------------------------------------------------------------------
#
#  Using DCOM :
#
#  To use EntireX DCOM with NaturalX a new Natural nucleus has to be linked.
#  This nucleus will support DCOM functionality.
#  In order to use the EntireX DCOM stubs please
#  specify the following on command line (in addition to other settings)
#
#     dco=yes
#
#  The following Environment variables must be set :
#
#    DCODIR     : EntireX DCOM base directory
#    DCOVERS    : EntireX DCOM version
#
#------------------------------------------------------------------------
#
#  The result of this make will be an executable 'natural'
#  which will be placed in the current directory
#
#  Use 'make install' for copying the generated 'natural' program
#  to $NATDIR/$NATVERS/bin. Original 'natural' program will be renamed
#  to 'natural.old'
#
##########################################################################
SHELL=/bin/sh
#
# Default Rule for Help text
#
ALL: default_text
#
##########################################################################
#
# Set own userexit files
#
USEROBJS = mqwrap.o jumptab.o
#
# Set include directory
#
INCDIR   = -I. -I$(NATDIR)/$(NATVERS)/samples/sysexuex -I /opt/mqm/inc
#
##########################################################################
#
# Set the database specific libraries
#
# ADABAS :
# --------
ada=
adaxvers=
LIB_DBXADA    = $(NATDIR)/$(NATVERS)/bin/build/dbxada$(adaxvers).a
LIB_ADA_STAT    = $(ADADIR)/$(ADAVERS)/adabas.a
LIB_ADA_DYN     = $(ADADIR)/$(ADAVERS)/adabas.o
#
# NET-WORK :
# ----------
```

```
LIB_CSCI_STAT   = $(NETDIR)/$(NETVERS)/lib/csci.a
LIB_CSCI_DYN    = $(NETDIR)/$(NETVERS)/lib/csci.o
LIB_BROKER_STAT = $(NETDIR)/$(NETVERS)/lib/broker.a
LIB_BROKER_DYN  = $(NETDIR)/$(NETVERS)/lib/broker.o
#
# ENTIRE :
# --------
ent=
entxvers=
LIB_DBXENT   = $(NATDIR)/$(NATVERS)/bin/build/dbxent$(entxvers).a
LIB_ENT         = $(AERDIR)/$(AERVERS)/lib/entdb.a
#
# SQL :
# -----
sql=
sqlxvers=
LIB_DBXSQL   = $(NATDIR)/$(NATVERS)/bin/build/dbxsql$(sqlxvers).a
LIB_SQL_LST         = $(OSQDIR)/$(OSQVERS)/osqlibs.lst
#
# OSX :
# -----
osx=
osxxvers=
LIB_DBXOSX   = $(NATDIR)/$(NATVERS)/bin/build/dbxsql$(osxxvers).a
LIB_OSX_LST         = $(OSXDIR)/$(OSXVERS)/osxlibs.lst
#
# ADABAS SQL server :
# -------------------
esq=
LIB_ESQ             = $(ESQDIR)/$(ESQVERS)/lib/esqint.o
#
# TP Monitor :
# ------------
tp=
tpxvers=
LIB_DBXTP    = $(NATDIR)/$(NATVERS)/bin/build/dbxtx$(tpxvers).a
LIB_TP_LST   = tplibs.lst
#
# DCOM :
# ------
#
dco=
LIB_DCO             = $(NATDIR)/$(NATVERS)/bin/natcom.a -L$(DCODIR)/$(DCOVERS)/lib
-loleaut32 -lole32 -lrpcrt4 -lntrtl -lmutant -lcoolmisc -L/usr/lib -lC -lw -
lpthread
#
#
# SyncSort :
# ----------
#
sync=
LIB_SYNC     = -lsyncsort
#
##########################################################################
#
# Set some general variables
#
CC     = /opt/SUNWspro/bin/cc
CFLAGS  =
LD     = /opt/SUNWspro/bin/cc
LFLAGS1 =
LFLAGS2 = -lm -lsocket -lnsl
LFLAGS3 = $(exx:yes=-lpthread)
#
# set name of temporary files used for linking
#
tmpdbexe=.dbexe
tmpdbext=.dbext
```

```
tmpsprod=.sprod
#
##########################################################################
#
# Set location of prelinked natural and some libraries
#
NATRAW       = $(NATDIR)/$(NATVERS)/bin/natraw.o
LIB_OW       = $(NATDIR)/$(NATVERS)/bin/build/libow.a
LIB_CM       = $(NATDIR)/$(NATVERS)/bin/build/libcm.a
LIB_RESOLV   = $(NATDIR)/$(NATVERS)/bin/build/libresolv.a
LIB_STUBCSCI = $(NATDIR)/$(NATVERS)/bin/build/stubcsci.a
LIB_STUBSYNC = $(NATDIR)/$(NATVERS)/bin/build/stubsync.a
PROLIB_LST   = natprolib.lst
MQLIBS       = -L /opt/mqm/lib   \
               -mt               \
               -lmqm             \
               -lmqmcs           \
               -lmqmzse          \
               -lsocket          \
               -lnsl             \
               -ldl
#
# Get directory for external libraries from Natural.INI file
#
NAT_EXTLIB   = grep NATEXTLIB $(NATDIR)/$(NATVERS)/etc/Natural.INI|sed 's/ //g'|sed
's/^.*=//'
#
##########################################################################
default_text:
	@echo '============================================================'
	@echo ' Natural link script                         (c) SOFTWARE AG'
	@echo '============================================================'
	@echo 'This Makefile is currently executing the default rule for'
	@echo 'giving you some help.'
	@echo ''
	@echo 'Use following command line for creating a new Natural in'
	@echo 'current directory:'
	@echo ''
	@echo '  make natural {<flags>}'
	@echo ''
	@echo 'For installation after successful creation call:'
	@echo ''
	@echo '  make install'
	@echo ''
	@echo 'Following Natural DB interface options {<flags>} are available:'
	@echo ''
	@echo '  ada=stat       link with static ADABAS interface'
	@echo '  ada=dyn        link with dynamic ADABAS interface'
	@echo '  ada=cscistat   link with static CSCI library also for ADABAS'
	@echo '  ada=cscidyn    link with dynamic CSCI library also for ADABAS'
	@echo ''
	@echo '  ent=yes        link with ENTIRE interface'
	@echo ''
	@echo '  sql=yes        link with SQL interface (OSQ)'
	@echo ''
	@echo '  osx=yes        link with SQL interface (OSX)'
	@echo ''
	@echo '  esq=yes        link with ADABAS SQL server interface'
	@echo ''
	@echo '  tp=yes         link with TP interface'
	@echo ''
	@echo '  sync=yes       link with SYNCSORT library'
	@echo ''
	@echo '  exx=yes        link with multithread library (used for Broker)'
	@echo ''
	@echo '  dco=yes        link with DCOM support'
	@echo ''
	@echo 'see top of Makefile for more information'
	@echo '============================================================'
	@echo ''

##########################################################################
#
# Rule to build a new natural
#
natural : db_all $(USEROBJS)
	@echo '---------------------------------------------------------------'
	@echo used db-interfaces:
	@cat $(tmpdbexe) $(tmpdbext)
	@echo '---------------------------------------------------------------'
	@echo used subproducts:
	@cat $(tmpsprod)
	@echo '---------------------------------------------------------------'
	@echo linking:
	$(LD) $(LFLAGS1)  \
	  $(NATRAW) \
	  $(LIB_OW) \
	  $(LIB_CM) $(USEROBJS) \
	  `cat $(tmpdbexe) $(tmpdbext) $(tmpsprod)` \
	  $(LIB_RESOLV) \
	  $(LFLAGS2) -m -ldl $(LFLAGS3) $(MQLIBS) -o natural
	@rm -f $(tmpdbexe) $(tmpdbext) $(tmpsprod)
	-rm -f $(USEROBJS)
	@echo 'linking complete'

#############################################################################
# dependency rules for DB interfaces
#############################################################################
db_all : db_start db_tp_$(tp) db_sql_$(sql) db_ent_$(ent) db_osx_$(osx) \
	db_esq_$(esq) db_ada_$(ada) db_sync_$(sync) db_dco_$(dco)

db_start :
	@-rm -f $(tmpdbexe) $(tmpdbext) $(tmpsprod)
	@-touch $(tmpdbexe) $(tmpdbext) $(tmpsprod)
	@-touch `$(NAT_EXTLIB)`/$(PROLIB_LST)
	@echo "echo \"`cat \`$(NAT_EXTLIB)\`/$(PROLIB_LST)`\"" | sh >> $(tmpsprod)

db_tp_yes :
	@-touch `$(NAT_EXTLIB)`/$(LIB_TP_LST)
	@echo $(LIB_DBXTP)  >> $(tmpdbexe)
	@echo "echo \"`cat \`$(NAT_EXTLIB)\`/$(LIB_TP_LST)`\"" | sh >> $(tmpdbext)

db_tp_ :

db_sql_yes :
	@-touch $(LIB_SQL_LST)
	@echo $(LIB_DBXSQL) >> $(tmpdbexe)
	@echo "echo \"`cat $(LIB_SQL_LST)`\"" | sh >> $(tmpdbext)

db_sql_ :

db_osx_yes :
	@-touch $(LIB_OSX_LST)
	@echo $(LIB_DBXOSX) >> $(tmpdbexe)
	@echo "echo \"`cat $(LIB_OSX_LST)`\"" | sh >> $(tmpdbext)

db_osx_ :

db_ent_yes :
	@echo $(LIB_DBXENT) >> $(tmpdbexe)
	@echo $(LIB_ENT)    >> $(tmpdbext)

db_ent_ :

db_esq_yes :
	@echo $(LIB_DBXADA) >> $(tmpdbexe)
	@echo $(LIB_STUBCSCI)	>> $(tmpdbext)
	@echo $(LIB_ESQ)	>> $(tmpdbext)
	@echo $(LIB_CSCI_DYN)	>> $(tmpdbext)
	@echo $(LIB_BROKER_STAT) >> $(tmpdbext)

db_esq_ :

db_ada_stat :
	@echo $(LIB_DBXADA) >> $(tmpdbexe)
	@echo $(LIB_ADA_STAT)	>> $(tmpdbext)

db_ada_dyn :
	@echo $(LIB_DBXADA) >> $(tmpdbexe)
	@echo $(LIB_ADA_DYN)	>> $(tmpdbext)

db_ada_cscistat :
	@echo $(LIB_DBXADA) >> $(tmpdbexe)
	@echo $(LIB_STUBCSCI)	>> $(tmpdbext)
	@echo $(LIB_CSCI_STAT)	>> $(tmpdbext)

db_ada_cscidyn :
	@echo $(LIB_DBXADA) >> $(tmpdbexe)
	@echo $(LIB_STUBCSCI)	>> $(tmpdbext)
	@echo $(LIB_CSCI_DYN)	>> $(tmpdbext)

db_ada_ :

db_sync_yes :
	@echo $(LIB_STUBSYNC)	>> $(tmpdbext)
	@echo $(LIB_SYNC)	>> $(tmpdbext)

db_sync_ :

db_dco_yes :
	@echo $(LIB_DCO)	>> $(tmpdbext)

db_dco_ :

#
# Use the following target to install the executable Natural.
# The one in the current directory is NOT removed!
# (call 'make install' at command line)
#
install:
	if [ -f $(NATDIR)/$(NATVERS)/bin/natural ] ; then \
	rm -f $(NATDIR)/$(NATVERS)/bin/natural.old ; \
	cp $(NATDIR)/$(NATVERS)/bin/natural $(NATDIR)/$(NATVERS)/bin/natural.old ; \
	fi
	cp natural $(NATDIR)/$(NATVERS)/bin/natural

#
# Use following target to clean up the current directory.
# (call 'make clean' at command line)
#
clean:
	-rm -f natural
	-rm -f *.o


#
# Rule to compile a .c file
#
.c.o:
	$(CC) $(CFLAGS) -c -O $(INCDIR) -DOS_UNIX=1 $*.c

#
```

# Appendix 2 The Natural Jumptable

```
*/
#include "nattab.h"
#include "natuser.h"
/*
** section 2: define external example routines
**
** ---------------------------------------------------------------------------
** START OF CODE TO BE CHANGED BY THE USER
*/
extern NATFCT NAMQCONN NATARGDCL(pcnt, pdat, pinf);
extern NATFCT NAMQDISC NATARGDCL(pcnt, pdat, pinf);
/*
** section 3: initialize the jumptable
**
** the function name visible to Natural must not be longer than 8
** uppercase characters.  they must contain only uppercase letters.
** the entries in the array must be alphabetically sorted by the
** function names.
*/
TAB_STRUCT n_call_table[] = {
  { "MQCONN", NAMQCONN } ,
  { "MQDISC", NAMQDISC }
};
/*
** section 4: make number of elements in the jump table availale to
** Natural.
**
** END OF CODE TO BE CHANGED BY THE USER
** ---------------------------------------------------------------------------
*/
TAB_SIZE n_call_tab = sizeof(n_call_table) / sizeof(n_call_table[0]);
/*
** end of file jumptab.c
*/
```

# Appendix 3 An example of the 'C' code for Connect and disconnect

```
/*****************************************************************************
**
** File:        mqwrap.c
** Purpose:     Example of MQ and Natural Interface
** Description: MQ functions
**
** (c) Copyright 2000 by IBM
**
*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>             /* Memory and type conversions */
#include <string.h>             /* strcpy stuff */
#include <ctype.h>              /* is upper stuff */
#include <time.h>               /* time of course */
#include <errno.h>              /* contains errno , last error indicator */
#include <fcntl.h>              /* FILE I/O related protos and defines */
#include <limits.h>             /* FILE I/O related protos and defines */
#include <unistd.h>             /* defines for file access              */
#include <sys/types.h>          /* for data types sycj as time          */
#include <sys/stat.h>           /* finding out status info              */
#include "cmqc.h"                /* MQ Headers       */
#include "natuser.h"             /* Natural User Exit      */
/*
** define function prototypes to avoid compiler warnings
*/
NATFCT NAMQCONN NATARGDCL(nparm, parmptr, parmdec);
NATFCT NAMQDISC NATARGDCL(nparm, parmptr, parmdec);
void mq_connect (MQHCONN * con_hndl, MQCHAR48 con_name, MQLONG *cc, MQLONG *rc);
void mq_disconnect (MQHCONN * con_hndl, MQLONG *cc, MQLONG *rc) ;

NATFCT NAMQCONN NATARGDEF(nparm, parmptr, parmdec)
{
  static char * func = "NAMQCONN : "    ;
             int    step                ;
  int i;                        /* loop counter */
  MQHCONN    con_hndl     ;
  MQCHAR48   con_name     ;
  MQLONG     cc           ;
  MQLONG     rc           ;
  /*
  ** test number of arguments
  */
  if (nparm != 4)
       {
             printf("\n%s Not ENough Parameters!\n" ,func) ;
             return 1;
       }
  /*
  ** test types of arguments
  for (i = 0; i < (int) nparm; i++)
      {
             if (parmdec[i].typevar != 'I' ||
                    parmdec[i].flen.lfield != sizeof(NATTYP_I4))
                    {
                          printf("\nParameter number %d is of incorrect type \n" ,
i ) ;
                    return 2;
                    }
       }
   */
  /*
  ** move Con name from Natural to Local variable
  */
  memmove ((char *) con_name, (char *) parmptr[0], sizeof(MQCHAR48));
  printf("\n%s con name is '%s'\n", func , con_name ) ;
  /*
  ** Call the connect routine
  */
  mq_connect ( &con_hndl, con_name, &cc, &rc);
  printf("\n%s handle = %d cc is %d , rc = %d \n", func , con_hndl , cc , rc ) ;
  /*
  ** move Handle to Natural Return block
  */
  memmove ((char *) parmptr[1], (char *) &con_hndl, sizeof(NATTYP_I4));
  /*
  ** move Condition Code to Natural Return block
  */
  memmove ((char *) parmptr[2], (char *) &cc, sizeof(NATTYP_I4));
  /*
  ** move Return Code to Natural Return block
  */
  memmove ((char *) parmptr[3], (char *) &rc, sizeof(NATTYP_I4));
  return 0;
} /* MQCON */
NATFCT NAMQDISC NATARGDEF(nparm, parmptr, parmdec)
{
  static char * func = "NAMQDISC : "    ;
             int    step                ;
  int i;                        /* loop counter */
  MQHCONN    con_hndl     ;
  MQLONG     cc           ;
  MQLONG     rc           ;
  /*
  ** test number of arguments
  */
  if (nparm != 3)
       {
             printf("\n%s Not Enough Parameters!\n" , func ) ;
             return 1;
        }
  /*
  ** test types of arguments
  for (i = 0; i < (int) nparm; i++)
      {
             if (parmdec[i].typevar != 'I' ||
                    parmdec[i].flen.lfield != sizeof(NATTYP_I4))
                    {
                           printf("\nParameter number %d is of incorrect type \n" ,
i ) ;
                    return 2;
                    }
       }
   */
  /*
  ** move Con handle from Natural to Local variable
  */
  memmove ((char *) &con_hndl, (char *) parmptr[0], sizeof(NATTYP_I4));
  printf("\n%s con handle = '%d' \n", func , con_hndl ) ;
  /*
  ** Call the Disconnect routine
  */
  mq_disconnect ( &con_hndl, &cc, &rc);
  printf("\n%s cc is %d , rc = %d \n", func, cc , rc ) ;
  /*
  ** move Handle to Natural Return block
  */
  memmove ((char *) parmptr[0], (char *) &con_hndl, sizeof(NATTYP_I4));
  /*
  ** move Condition Code to Natural Return block
  */
  memmove ((char *) parmptr[1], (char *) &cc, sizeof(NATTYP_I4));
  /*
  ** move Return Code to Natural Return block
  */
  memmove ((char *) parmptr[2], (char *) &rc, sizeof(NATTYP_I4));
  return 0;
} /* NAMQDISC */
/*
***********************************************************************
                    mq_connect
***********************************************************************
*/
void mq_connect (MQHCONN * con_hndl, MQCHAR48 con_name, MQLONG *cc, MQLONG *rc)
{
  static char * func = "mq_connect : "  ;
             int    step                ;
#ifdef MQ_DEBUG
       printf("in %s\n" , func ) ; getchar() ; getchar() ;
#endif
    MQCONN(con_name, con_hndl, cc , rc) ;
    if ( (*cc == MQCC_WARNING ) && ( *rc == MQRC_ALREADY_CONNECTED) )
       {
         printf("%s Already connected .. Warning ignored by program\n",func);
         *cc = MQCC_OK ;
       }
    if ( *cc == MQCC_FAILED)
       {
          printf("%s Failed to Connect\n",func);
       }
    return ;
}
/*
***********************************************************************
             mq_disconnect
***********************************************************************
*/
void mq_disconnect (MQHCONN * con_hndl, MQLONG *cc, MQLONG *rc)
{
  static char * func = "mq_disconnect : "      ;
             int    step                ;
#ifdef MQ_DEBUG
       printf("in %s\n" , func ) ; getchar() ; getchar() ;
#endif
     MQDISC(con_hndl, cc, rc);
     if ( *cc  != MQCC_OK)
       {
             printf("\n%s cc is %d , rc = %d \n", func, cc , rc ) ;
       }
    return ;
}
```

# Appendix 4: Examples of MQSeries Data Structure definitions for Natural

## MQSeries PutMessage Options

```
        1 MQPMO                                  A  128
R      1 MQPMO                                 /* REDEF. BEGIN : MQPMO
        2 MQPMO-STRUCID                          A    4
        2 MQPMO-VERSION                           I    4
        2 MQPMO-OPTIONS                           I    4
        2 MQPMO-TIMEOUT                           I    4
        2 MQPMO-CONTEXT                           I    4
        2 MQPMO-KNOWNDESTCOUNT                  I    4
        2 MQPMO-UNKNOWNDESTCOUNT                I    4
        2 MQPMO-INVALIDDESTCOUNT                  I    4
        2 MQPMO-RESOLVEDQNAME                    A   48
        2 MQPMO-RESOLVEDQMGRNAME               A   48
```

## MQSeries GetMessage Options

```
          1 MQGMOV
            2 MQGMO-STRUCID                A    4 INIT<'GMO'>
           2 MQGMO-VERSION                 I    4 INIT<1>
           2 MQGMO-OPTIONS                 I    4
           2 MQGMO-WAITINTERVAL            I    4
           2 MQGMO-SIGNAL1                  I    4
           2 MQGMO-SIGNAL2                  I    4
           2 MQGMO-RESOLVEDQNAME       A   48
```

## MQSeries Close Call Option Values

```
          1 MQCO-NONE                   I    4 INIT<0>
          1 MQCO-DELETE                 I    4 INIT<1>
          1 MQCO-DELETE-PURGE        I    4 INIT<2>
```

## MQSeries Object Descriptor

```
         1 MQOD                             A  168
R     1 MQOD                             /* REDEF. BEGIN : MQOD
       2 MQOD-STRUCID                    A    4
          2 MQOD-VERSION                    I    4
          2 MQOD-OBJECTTYPE                 I    4
          2 MQOD-OBJECTNAME                A   48
          2 MQOD-OBJECTQMGRNAME       A   48
          2 MQOD-DYNAMICQNAME           A   48
          2 MQOD-ALTERNATEUSERID        A   12
```

## MQSeries Open Option Variables

```
         1 MQOO-INPUT-AS-Q-DEF                     I    4 INIT<1>
         1 MQOO-INPUT-SHARED                       I    4 INIT<2>
         1 MQOO-INPUT-EXCLUSIVE                    I    4 INIT<4>
         1 MQOO-BROWSE                             I    4 INIT<8>
         1 MQOO-OUTPUT                             I    4 INIT<16>
         1 MQOO-INQUIRE                            I    4 INIT<32>
         1 MQOO-SET                                I    4 INIT<64>
         1 MQOO-SAVE-ALL-CONTEXT                   I    4 INIT<128>
         1 MQOO-PASS-IDENTITY-CONTEXT              I    4 INIT<256>
         1 MQOO-PASS-ALL-CONTEXT                   I    4 INIT<512>
         1 MQOO-SET-IDENTITY-CONTEXT               I    4 INIT<1024>
         1 MQOO-SET-ALL-CONTEXT                    I    4 INIT<2048>
         1 MQOO-ALTERNATE-USER-AUTHORITY           I    4 INIT<4096>
         1 MQOO-FAIL-IF-QUIESCING                  I    4 INIT<8192>
```

## MQSeries Completion Codes

```
         1 MQCC-OK                    I    4 INIT<0>
         1 MQCC-WARNING               I    4 INIT<1>
         1 MQCC-FAILED                I    4 INIT<2>
         1 MQCC-UNKNOWN               I    4 INIT<-1>
```

---

# MQSeries Message Descriptor

```
1 MQMDV
 2 MQMD-STRUCID                A   4 INIT<'MD  '>
 2 MQMD-VERSION                I   4 INIT<1>
 2 MQMD-REPORT                 I   4 INIT<0>
 2 MQMD-MSGTYPE                I   4 INIT<8>
 2 MQMD-EXPIRY                 I   4 INIT<-1>
 2 MQMD-FEEDBACK               I   4 INIT<0>
 2 MQMD-ENCODING               I   4 INIT<785>
 2 MQMD-CODEDCHARSETID        I   4 INIT<0>
 2 MQMD-FORMAT                 A   8 INIT<' '>
 2 MQMD-PRIORITY               I   4 INIT<-1>
 2 MQMD-PERSISTENCE            I   4 INIT<2>
 2 MQMD-MSGID                  A  24 INIT FULL LENGTH<H'00'>
 2 MQMD-CORRELID               A  24 INIT FULL LENGTH<H'00'>
 2 MQMD-BACKOUTCOUNT           I   4 INIT<0>
 2 MQMD-REPLYTOQ               A  48 INIT<' '>
 2 MQMD-REPLYTOQMGR            A  48 INIT<' '>
 2 MQMD-USERIDENTIFIER         A  12 INIT<' '>
 2 MQMD-ACCOUNTINGTOKEN       A  32 INIT FULL LENGTH<H'00'>
 2 MQMD-APPLIDENTITYDATA      A  32 INIT<' '>
 2 MQMD-PUTAPPLTYPE            I   4 INIT<0>
 2 MQMD-PUTAPPLNAME            A  28 INIT<' '>
 2 MQMD-PUTDATE                A   8 INIT<' '>
 2 MQMD-PUTTIME                A   8 INIT<' '>
 2 MQMD-APPLORIGINDATA         A   4 INIT<' '>
```

**End of Document**