Middleware Suite - Application Integration

NCI - Network Computing Interface

MQ File Transfer Utilities
Implementation Guide
Version: 3.1

T · · Systems ·

Impressum

Publisher	**T-Systems Enterprise Services GmbH** Computing Services & Solutions (CSS) System Products & Automation (MSY-PA)
Responsible	**T-Systems Enterprise Services GmbH** Computing Services & Solutions (CSS) System Products & Automation (MSY-PA) Fasanenweg 9, 70771 Leinfelden-Echterdingen, Germany cc.middleware@t-systems.com +49 89/1011-4687

Document information

Name of file

NCI MQ File Transfer Utilities

Version / Revision	Date	Revision
This edition relates to NCI version **NCI 3.1**	**25/11/2005 08:49:00**	**220**

- **PNCI310/QZ05046** on z/OS
- **PNCI310/REL1003** on Unix/Windows

List of available NCI documentation:

NCI Application Programming Reference
NCI Installation and Customization for Distributed Systems
NCI Installation and Customization for z/OS and OS/390
NCI MQ File Transfer Utilities
NCI MQ Security Suite
NCI SAP R/3 RFC Server Interface
NCI Additional Features

Additional License Information

Acknowledgment:

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (http://www.openssl.org/)
This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)
This product includes software written by Tim Hudson (tjh@cryptsoft.com)
This product includes software developed by the Apache Software Foundation (http://www.apache.org/)

Table of Contents

List of Figures

1 Overview

NCI provides two utilities, as well as a documented interface, which allow the transfer of files via MQSeries.

The utility *NCIF2Q* allows the transformation of flat files into several MQSeries messages, that will be transmitted, together with appropriate meta-information via an MQSeries network to another location, where the messages will be re-assembled by *NCIQ2F* into the original file while taking account of platform-specific codepages and record structure. The utilities can be configured via command-line parameters and a sideinformation file.

The layout and content of the meta-information message, as well as the sequence of the messages composing a file, are documented, in order to allow EAI adapters to be build, that conform to the specification.

The utilities process file transfer requests according to the following specifications:

- The configuration will be done via command-line parameters and a sideinformation file.
- At the sending site, a file is split into a number of blocks of a configurable size and each block is written to an MQSeries queue defined in the sideinformation file under a symbolic name. The symbolic name will be specified as a command-line parameter to *NCIF2Q*. Meta information, which allows the re-assembly of the file while preserving its integrity is added in the form of extra messages and MQSeries message header fields.
- At the receiving site, the file is re-assembled only when the last block has been received and when all blocks belonging to a file are available. The file will only be written, when it is complete.
- The record structure of the file is preserved, depending on configuration parameters in the sideinformation file.
- Transmission of the messages composing the file is performed by MQSeries. It is the responsibilty of the MQSeries administrator to ensure that all messages reach their destination, i.e. starting of channels in the case of transmission errors or moving of messages from application-specific dead-letter queues back to the receiving queue if required. The receiving queue will likewise be specified in the sideinformation file of *NCIQ2F* and belongs to a symbolic name.
- A log message will be written at every important event, i.e. a file has been written to disk or a follow-up message has been put to a queue.
- Follow-up processes are supported on either site of a file transfer request. They can be started either by using the standard MQ triggering mechanism or by a manually started program.
    - At the sending site the trigger for follow-up processes is the arrival of a reply message in a predefined reply-to queue from the receiver of the file, i.e. *NCIQ2F* and/or a user-written follow-up program at the receiving site.
    - At the receiving site the trigger for follow-up processes is a follow-up message written to a predefined follow-up queue after the file has been written to disk.
    - The reply message and follow-up messages are written in the same pre-defined XML-style format, that can be processed by a user application.
- Reply messages will be sent by *NCIQ2F* only when the file has been written to disk successfully. In the case of an error at the receiving site, no reply message will be sent, because it is assumed that recovery from the error is possible. Reply messages by *NCIQ2F* can also be suppressed by a sideinformation parameter , if desired. A user-written follow-up program can send another reply message, in order to implement end-to-end control.
- *NCIQ2F* tries to recover from errors a configurable number of times in configurable time periods. If this was unsuccessful, the messages belonging to the file will be moved to an application-specific dead-letter queue and a log message will be written. If the messages will be put back to the receiving queue at a later time by an administrator or a dead-letter queue-handler processing continues as before.
- In order to identify a file transfer request in a reply or a follow-up message, a request number can be specified as a command-line parameter to *NCIF2Q*.

- An application-specific dead-letter queue is required. *NCIQ2F* will not start if this queue is missing in the sideinformation file. Messages in the application-specific dead-letter queue will be written with a standard MQDLH (Dead-letter header).
- The utilities can be started by command with appropriate parameters or by job or script respectively. Additionally *NCIQ2F*, which retrieves the messages composing a file and writes the file to disk, can be started directly by MQSeries triggering or indirectly by AJM which itself gets triggered by MQSeries triggering.

The configuration parameters and the specification of the message structure will be described in detail in later chapters.

2 Quick Start

In the following the steps are described in short, that are required to carry out a file transfer with the *NCI MQ File Transfer Utilities* across an MQSeries network. For details refer to the appropriate chapters in this guide and in the MQSeries documentation.

1. Define the required MQSeries objects for intercommunication according to the sample provided on "Sample MQSeries Definitions for Intercommunication" on page 30

   - Define a transmission queue at the sending site

     Specify message delivery sequence as FIFO

   - Define a sender channel at the sending site
   - Define a trigger process for the sender channel
   - Define a receiver channel at the receiving site

2. Define the required MQSeries objects for the file transfer according to the sample provided on "Sample MQSeries definitions" on page 28

   - Define a queue manager alias for the sending queue manager at the receiving site
   - Define a local queue at the receiving site for receiving files

     Specify message persistence as YES

   - Define a remote queue at the sending site that refers to the receive queue
   - Define a local queue at the receiving site for follow-up messages

     Specify message persistence as YES

   - Define a local queue at the sending site for reply messages

     Specify message persistence as YES

3. Create the sideinformation file at the sending site according to the sample provided on "Sample Sideinformation File for the Sending Site" on page 31

   - Use the example provided in this documentation as a starting point

4. Create the sideinformation file at the receiving site according to the sample provided on "Sample Sideinformation File for the Receiving Site" on page 32

   - Use the example provided in this documentation as a starting point

If *NCIQ2F* should be started manually in order to receive files, continue with the following steps:

5. Transmit a file with NCIF2Q

   - Specify the command-line parameters **-c <sideinfo> -s <symbname> -f <sourcefile>** where *sideinfo* denotes the name of the sideinformation file or module, *symbname* denotes the name of the symbolic section in the sideinformation and *sourcefile* denotes the name of the file, that has to

be transmitted.

6. Receive a file and write it to disk with NCIQ2F

- Specify the command-line parameters **-c <sideinfo> -s <symbname>** where *sideinfo* denotes the name of the sideinformation file or module and, *symbname* denotes the name of the symbolic section in the sideinformation.

If triggering should be used for starting *NCIQ2F* when a file arrives at the receiving site continue after point 4 above with the following steps:

5. Make the required MQSeries definitions according to "Sample MQSeries definitions" on page 28

- Configure the trigger monitor

  On z/OS the trigger monitor is implemented as a task of the NCI Communication Manager. See the *NCI Installation and Customization Guide* for details.

- Define a MQ PROCESS

  Specify the application id as **NCIQ2F** on distributed platforms and as **NCIQ2FT** on z/OS.
  On z/OS specify environment data as TASK

- Configure the receiving queue for triggering

  Set TRIGGER
  Set the trigger type to FIRST
  Set the trigger data to *-c <sideinfo> -s <symbname>*, where *sideinfo* denotes the name of the sideinformation file or load module and *symbname* denotes the name of the symbolic section in the sideinformation.
  Specify the trigger message priority as **1**

6. Start the trigger monitor
7. Transmit a file with NCIF2Q

- Specify the command-line parameters **-c <sideinfo> -s <symbname> -f <sourcefile>** where *sideinfo* denotes the name of the sideinformation file or module, *symbname* denotes the name of the symbolic section in the sideinformation and *sourcefile* denotes the name of the file, that has to be transmitted.

3 Interface Reference

3.1 NCIF2Q - Transformation of a file into MQ Messages

NCIF2Q allows the transformation of a flat file into a group of MQSeries messages, that will be put to a remote queue in order to be transmitted via an MQSeries network.

NCIF2Q has the following features:

Configurable via sideinformation and/or command-line parameters

Symbolic addressing of queues and queue managers

Tracing, logging, error-handling

Configurable maximum message size

Optional allocation parameters for the receiving site

The command syntax is as follows:

```
NCIF2Q -c sideinformation file -s SymbolicName
          [ -f SourceFileName] [-r RequestNumber]
          [ -l LogicalFileName] [-d DestinationSymbolicName]
          [ -u UserText] [-o DsOrg] [-e LRecl] [-t RecFm]
```

Figure 1: Syntax of NCIF2Q command

where:

- sideinformation file specifies the path and name of the sideinformation file. This parameter is required.
- SymbolicName specifies the *SymbolicName* entry in the sideinformation file. This parameter is required.
- SourceFileName specifies the name of the file to be transmitted. This parameter is optional. If omitted, the SourceFileName specified in the sideinformation file in the specified SymbolicNamesection will be used.
- RequestNumber specifies a user-defined file transfer request number, that will be included in reply and follow-up messages. This parameter is optional.
- LogicalFileName specifies the file name which will be used to create the target file name at the receiving site. The target file name will be constructed by appending the logical file name to the target directory specified in the sideinformation file at the receiving site.
- DestinationSymbolicName specifies the symbolic name in the sideinformation file at the receiving site to be used for parameter substition of parameters from the primary symbolicname at the receiving site.
- UserText allows to associate a string of a maximum length of 256 byte with the file transfer request. This string will be carried forward in the XML meta information to the reply queue and follow-up queue.
- DsOrg the data set organization to be used for the file at the receiving site. This parameter is only applicable if the target system is Tandem.

- **LRecl** the logical record length to be used for the file at the receiving site. This parameter is only applicable if the target system is z/OS or Tandem.
- **RecFm** the record format (FB of VB) to be used for the file at the receiving site. This parameter is only applicable if the target system is z/OS or tandem.

Note: *If a parameter can be specified on the command line as well as in the sideinformation file, the parameter from the command line takes precedence.*

3.2 NCIQ2F - Re-assembly of a file from MQ messages

The utility *NCIQ2F* allows the re-assembly of a group of MQSeries messages, retrieved from a configurable queue, into a flat file

NCIQ2F combines messages from a queue into a file in such a way, that the original file will be restored. The logical record structure will be preserved. Code page conversion will be performed if required.

NCIQ2F has the following features:

- Configurable via sideinformation and/or command-line parameters
- Symbolic addressing of queues and queue managers
- Tracing, logging, error-handling configurable
- Logical record structure preserved

The command syntax is as follows

```
NCIQ2F -c sideinformation file -s SymbolicName
```

Figure 2: Syntax of NCIQ2F command

where:

- **sideinformation file** specifies the path and name of the sideinformation file. This parameter is required.
- **SymbolicName** specifies the *SymbolicName* entry in the sideinformation file to be used exclusively by *NCIQ2F*. This parameter is required.

If *NCIQ2F* should be started by a trigger monitor, its parameter string has to be specified in the **TRIGDATA** field of the application queue definition.

In this case, it will be checked, if the application queue, which triggered the start of *NCIQ2F* is identical to those specified in the sideinformation file under the symbolic name specified in the parameter string. If the queues are different, *NCIQ2F* terminates with an error condition.

Note: *On z/OS specify **NCIQ2FT** instead of **NCIQ2F** in the MQ PROCESS definition when NCIQF2 should be started by a trigger monitor.*

3.3 Format and syntax of the sideinformation file

3.3.1 General syntax information

NCI sideinformation can be used to provide application specific control options, object names and addressing information outside the application.

The NCI sideinformation file contains a list of keywords (parameters) as described in the following. Each keyword must begin and end on a separate line. All data between the opening and the closing bracket is treated as the keyword value. Leading or trailing blanks are not stripped!

Note: *Keywords are not case sensitive. However the keyword values are all case sensitive. To prevent errors, code the keyword values in exactly the same syntax they are described in the reference chapters.*

Lines starting with a '*' are treated as comment lines.

Each NCI sideinformation file must contain at least one named symbolic section (keyword SymbolicName). NCI doesn't support a default section. A sideinformation file may contain multiple sections, offering the flexibility to define different application definitions within the same sideinformation file.

All keywords defined in the sideinformation file before the first symbolic section name are treated as global keywords. Global keywords are propagated to all subsequent symbolic sections.

If a global keyword is also defined within a symbolic section, the global definitions will be overridden by the definitions in the symbolic section.

The SideApplData keyword can appear multiple times with different sub-keywords in a symbolic section or in the global section. They are considered as one keyword with regard to the above, that is, no propagation of SideApplData keywords from the global section to a symbolic section occurs, if at least one SideApplData keyword has been specified in the symbolic section.

The symbolic name specified via the *NCIQ2f* command-line or receive queue definition, if MQSeries triggering will be used, is called the primary symbolicname. The initiator of a file transfer request has the option to specify a destination symbolic name in his sideinformation file or via the command-line of *NCIF2Q*. This destination symbolicname will be transmitted via XML-metainformation to the receiving site, where the corresponding symbolic section must be defined and will be used for parameter substitution according to the following rules:

- SideApplData keywords specified in the destination symbolic section override the corresponding SideApplData keywords in the primary symbolic section, with a few exceptions (see the keyword syntax for details).
- It is also possible to nullify some primary symbolic section keywords by specifying the parameter NONE for the corresponding destination symbolic section keyword.

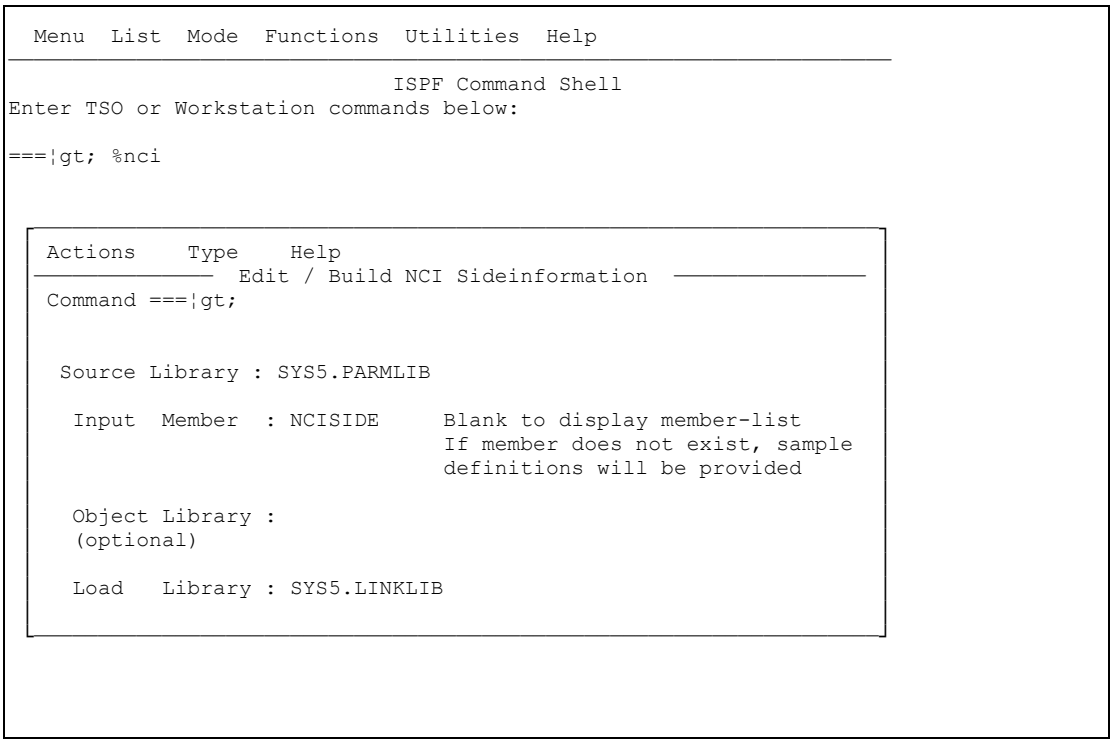
Other keywords, apart from SideApplData are ignored, for a symbolic section when being addressed by a destination symbolic name.

```
*                          // begin of global section
    AddrType(MQ)             // adddressing type: message queue
    PrimAddrInfo(QMGR1)      // default queue manager name
    SecAddrInfo(DFLTQ)       // default queue name
    TraceOpt(NONE)           // no tracing
    ErrorMsgOpt(STDERR)      // error messages will be written
    SideApplData(MaxMsgSize(4k)) // maximal message size
    SideApplData(Action(KEEP))   // delete file after tarnsformation

  SymbolicName(X1)           // begin of Symbolic section
    SecAddrInfo(Q1)          // queue name
    ErrorMsgOpt(FILE)        // error messages written to file
    ErrorMsgFile(/var/nci/error.log  // error message file
    SideApplData(FileName(/data/Y1)) // file name to be transformed

  SymbolicName(X2)           // Begin of Symbolic section
    SecAddrInfo(Q2)          // queue name
    TraceOpt(ALL)            // tracing required
    SideApplData(FileName(/data/Y2)) // file name to be transformed

  SymbolicName(X3)           // Begin of Symbolic section
    SecAddrInfo(Q3)          // queue name
    SideApplData(MaxMsgSize(32k))    // maximal message size
    SideApplData(FileName(/data/Y3)) // file name to be transformed
    SideApplData(Action(DELETE)      // delete file after tarnsformation
```

Figure 3: Example NCI sideinformation file

3.3.2 Creating a sideinformation module on z/OS

On z/OS the sideinformation file must be converted to a sideinformation load module. An ISPF-Dialog will assist you in generating a NCI sideinformation module. !



3.3.3 Sideinformation Keywords

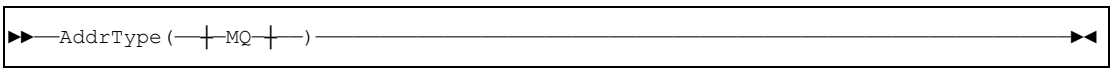


Figure 4: Addressing Type

Note: *This parameter is mandatory.*

*The parameter **AddrType** defines the transport protocol to be used.* The following addressing types are supported:

• Type	• Description
• MQ	• MQSeries will be used as transport protocol. The parameters PrimAddrInfo and SecAddrInfo will be used to address the MQSeries queue manager and queue name. PrimAddrInfo must provide the name of the MQSeries queue manager, otherwise the default queue manager will be used. SecAddrInfo must provide the MQSeries queue name.

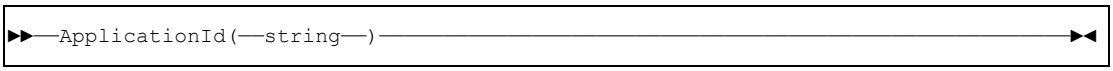


Figure 5: Application Identifier

The application identifier is used to identify the originator of an NCI error message. NCI error messages always include the Application Identifier.

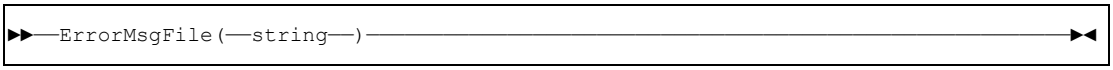


Figure 6: Error Message File

Note: This function is not supported on OS/390.

The default value for *ErrorMsgFile* is **ncierror.log**.
Defines the name of the file, where NCI error messages will be written if *ErrorMsgOpt(**FILE**)* is in effect.

Controls the way error messages will be handled by NCI. The application can access NCI error messages (independent from *ErrorMsgOpt*) via function *NCIGetErrorMsgText*.

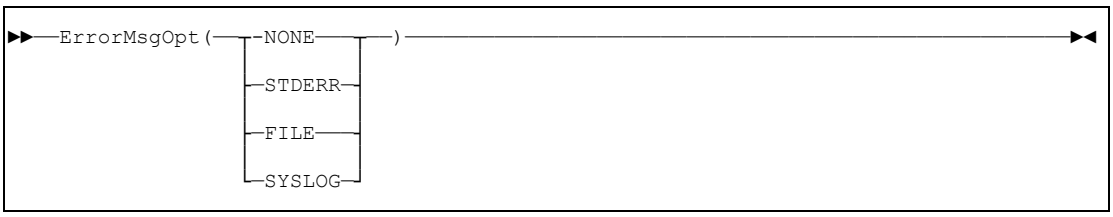


Figure 7: Error Message Processing Options

Value	Description
NONE	Error messages will be suppressed.
STDERR	Error messages will be written to STDERR (on z/OS: SYSLOG).
FILE	Error messages will be written to an application-specific error message file specified by the NCISetErrorMsgFile function call or via NCI sideinformation.
	Note This option is not supported for z/OS.
SYSLOG	Error messages will be written to the system log.
	Note This option is not supported for Windows

Specifies a period of time expressed in tenths of a second. The messages belonging to a file become

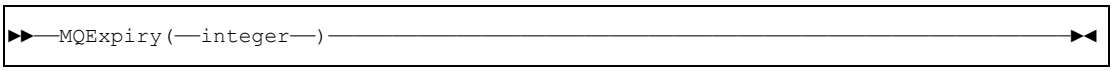


Figure 8: MQ Message Expiration Period

eligible to be discarded if they have not been removed from the destination queue before this time period has elapsed.

PrimAddrInfo specifies the name of the MQseries queue manager, to which the application should connect. If *PrimAddrInfo* is omitted or specified as blank, the application tries to connect to the default queue manager.

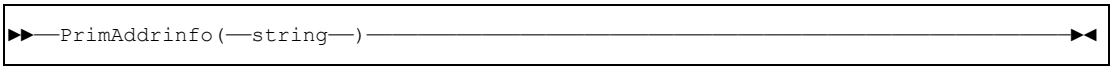


Figure 9: Primary Addressing Information

In case of a connection error, the retry number value specifies how many times a retry to establish the connection will be done.
A value of 0 means no retry. A value of 255 means unlimited number of retries.

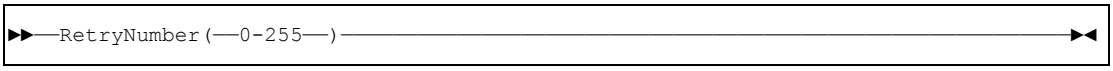


Figure 10: Retry Number

Connection errors for protocol MQSeries can occur if:

- the local queue manager is not active
- the requested host is not up (MQ client connection only)
- the network is unavailable (MQ client connection only)

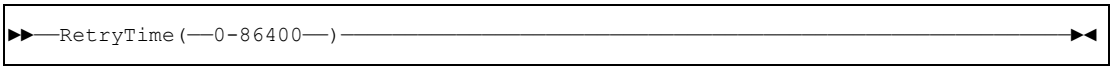


Figure 11: Retry Time

Specify a retry time value in seconds. If a connection error occurs, the *RetryTime* value will be used as wait time until the next attempt to establish a connection is started.

Connection errors for protocol MQSeries can occur if:

- the local queue manager is not active
- the requested host is not up (MQ client connection only)

- the network is unavailable (MQ client connection only)

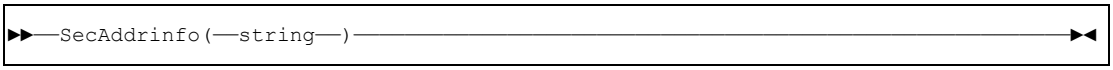


Figure 12: Secondary Addressing Information

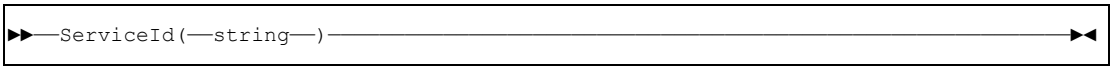


Figure 13: Service Identifier

SecAddrInfo specifies the queue name to be used for transformation from a file into MQSeries messages. The default value for *SecAddrInfo* is SYSTEM.DEFAULT.LOCAL.QUEUE.

With the parameter *ServiceId* the initiator of a file transfer request, i.e. *NCIF2Q*, can assign an identifier to all the messages belonging to the file.

The receiving application, i.e *NCIQ2F*, can use the parameter *ServiceId* as a search argument. Only messages with this specific *ServiceId* will be retrieved. *ServiceId* can be specified generic, i.e. with an asterisk as the last character.

ServiceId can have a maximum length of 28 bytes.

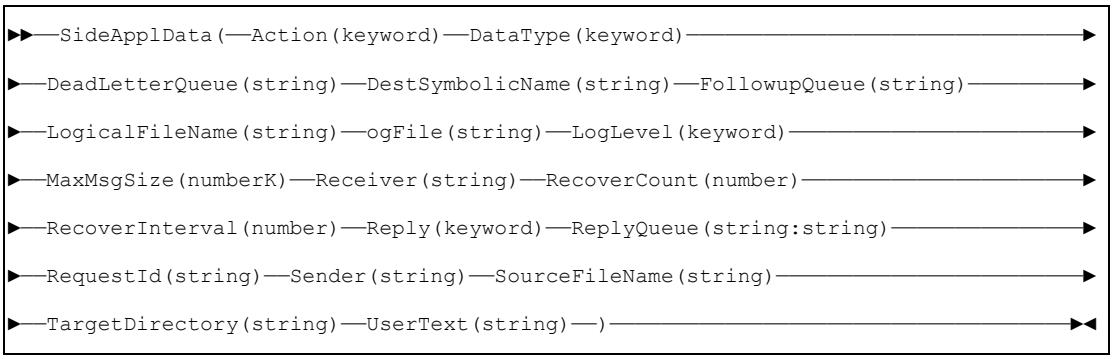


Figure 14: Sideinformation Application Data

The sideinformation application data controls the behaviour of the NCI file transfer utilities. Thekeywords are applicable either to *NCIF2Q* or *NCIQ2F* or both and are described in the next chapter.

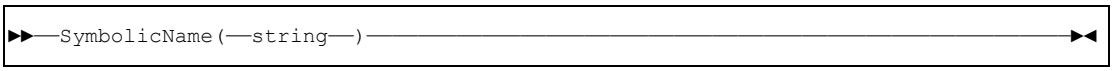


Figure 15: Symbolic Name

The *SymbolicName* will be used by an application

- as a logical name to address a physical target system/application (e.g. MQSeries queue manager/queue name).
- to provide applicationspecific control options (e.g. Timeout values / Trace options / ...) outside the application.

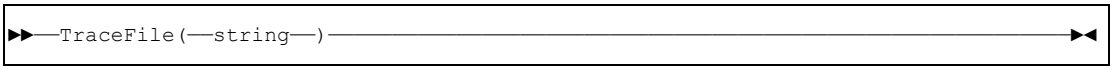


Figure 16: Trace File

Note: This function is not supported for operating system OS/390.

The default value for *TraceFile* is **ncitrace.log**.
Defines the name of the file, where NCI trace records will be written if tracing is in effect. Tracing can be activated by function call *NCISetTraceOpt* or via NCI sideinformation.

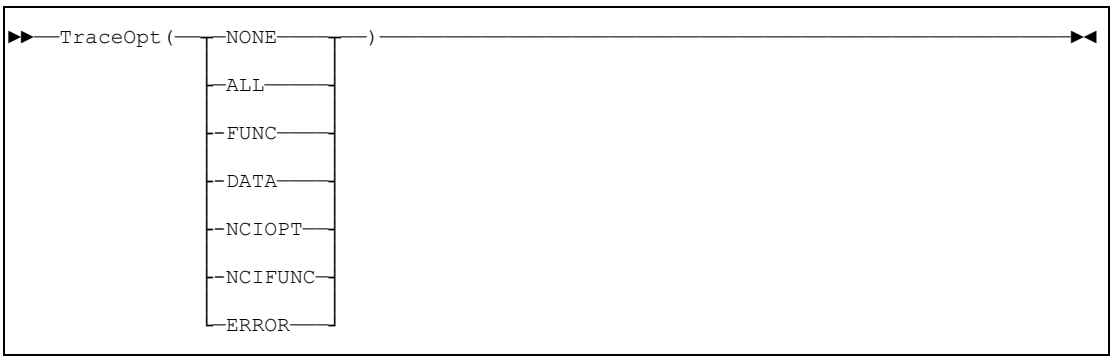


Figure 17: Trace Options

Note: Tracing is not supported on OS/390.

Setup Trace Options.

• Option	• Description
• NONE	• Disable Trace.
• ALL	• Trace all.
• FUNC	• Function Trace (MQSeries calls).
• NCIOPT	• NCI options being used.

• NCIFUNC	• NCI Function Trace (internal functions).
• ERROR	• Error messages are additionally written to the NCI tracefile.

3.3.4 SideAppl Data Keywords

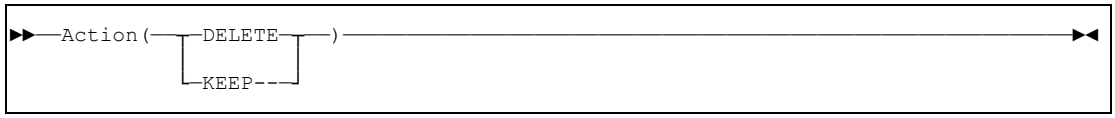


Figure 18: Action

Applicable only to *NCIF2Q*.

Specifies the disposition of the file to be transmitted.

DELETE The file will be deleted, if it has been successfully passed to MQSeries.

KEEP The file will be kept. This is the default value.

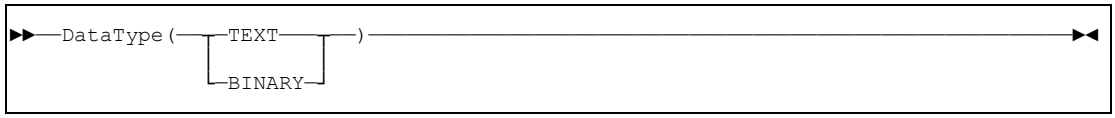


Figure 19: Data Type

Applicable only to *NCIF2Q*.

Specifies the typ of file to be transmitted.

TEXT Text file. Conversion will be done by MQSeries routines.

BINARY Binary file. No conversion will be done.

Note: Note, that binary files are currently not supported on Compaq NonStop Kernel (Tandem)

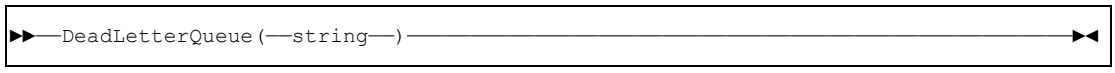


Figure 20: DeadLetterQueue

Applicable only to *NCIQ2F*. This parameter is required.

If a file cannot be written, because of an error situation, such as missing messages, after a given number of recover attempts, the messages belonging to the file will be written in standard MQSeries dead-letter queue format to the queue specified with the :hp1DeadLetterQueue.

Note: *The string supplied for the* ***DeadLetterQueue*** *keyword must be a* ***SymbolicName*** *entry defined in the same sideinformation file.*

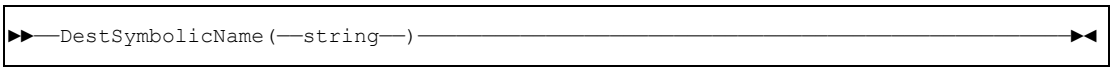


Figure 21: DestSymbolicName

Applicable only to *NCIF2Q*.

Specifies the destination symbolic name, i.e the name of the symbolic section at the receiving site, whose SideApplData keywords substitute the corresponding keywords of the primary symbolic section.

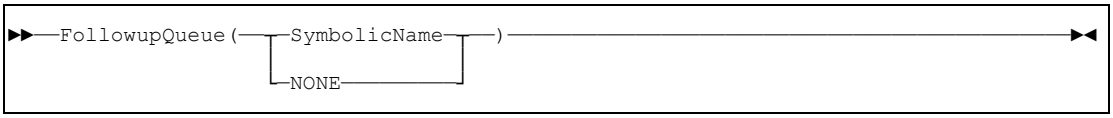


Figure 22: FollowupQueue

Applicable only to *NCIQ2F*.

When the messages composing a file have been written to disk at the receiving site, an MQSeries message containing information required for follow-up processes will be written to the FollowupQueue in a documented format. If the value NONE has been specified for the FollowupQueue, the following applies:

- If the symbolic section will be used as a primary symbolic section, spefifying NONE is equivalent to ommitting the FollowupQueue keyword.
- If the symbolic section will be used as a destination symbolic section, the FollowupQueue definition of the primary symbolic section will be nullified.

Note: The symbolic name specified with the FollowupQueue keyword must refer to a symbolicname section defined in the same sideinformation file.

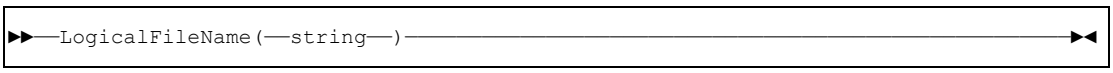


Figure 23: LogicalFileName

Applicable only to *NCIF2Q*.

The logical file name is used to construct the target file name at the receiving site.

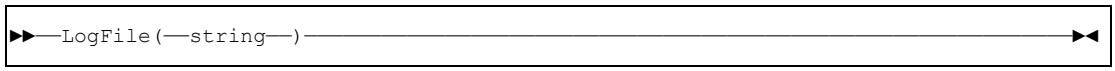


Figure 24: LogFile

Applicable to *NCIF2Q* and *NCIQ2F*.

Specifies the name (including path information) of the file, where messages will be written.

For a destination symbolic name this keyword will be ignored.

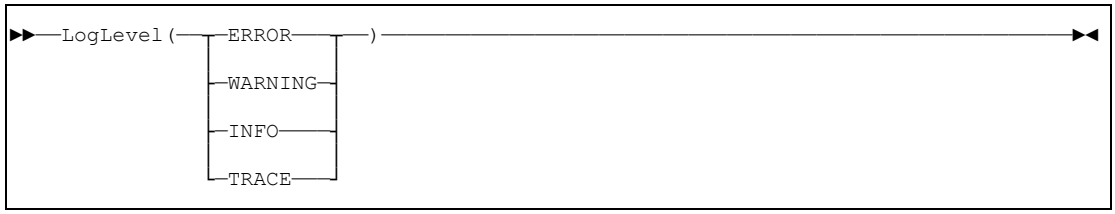


Figure 25: LogLevel

Applicable to *NCIF2Q* and *NCIQ2F*.

Specifies which kind of messages will be written to the log file.

ERROR	Only error messages will be written.
WARNING	Error messages and warnings will be written
INFO	Error messages, warnings and info messages will be written
TRACE	All messages, including trace messages, will be written

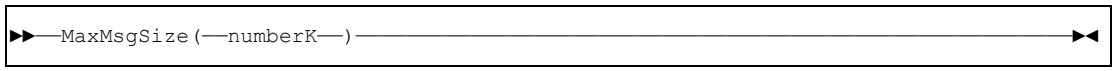


Figure 26: MaxMsgSize

Applicable only to *NCIF2Q*

Specifies the maximum message size in KByte, that is used, when a file on the sending site is split into blocks for transmission via MQSeries. The default value is **64K**.

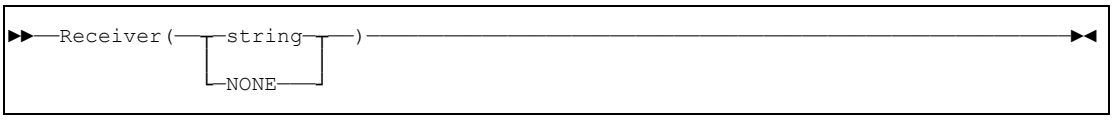


Figure 27: Receiver

Applicable to *NCIF2Q* and *NCIQ2F*.

Specifies the name of the receiving system. If the value **NONE** has been specified for the Receiver, the following applies: This parameter has no significance regarding the transmission of the file. The value specified will be contained in all meta-information messages

- If the symbolic section will be used as a primary symbolic section, spefifying NONE is equivalent to ommitting the Receiver keyword.
- If the symbolic section will be used as a destination symbolic section, the Receiver definition of the primary symbolic section will be nullified.

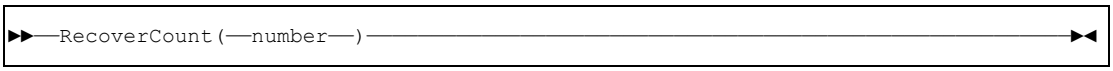


Figure 28: Recover Count

Applicable only to *NCIQ2F*.

Specifies the number of retries that are done, if *NCIQ2F* fails to compose a file from MQSeries messages in the case of recoverable errors, for example because messages are missing.

A value of 0 means no retry. A value of 255 means unlimited number of retries.

For a destination symbolic name this keyword will be ignored.

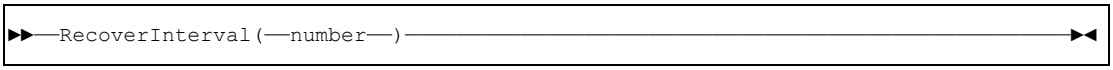


Figure 29: RecoverInterval

Applicable only to *NCIQ2F*.

Specifies the time period in minutes between successive recovery attempts in the case of recoverable errors.

For a destination symbolic name this keyword will be ignored.

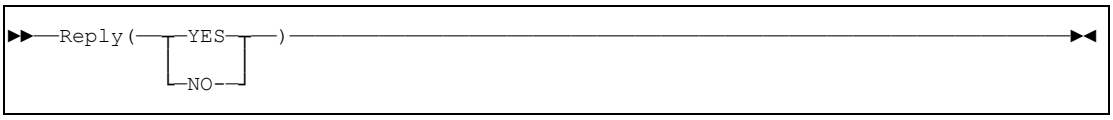


Figure 30: Reply

Applicable only to NCIQ2F. The default is YES.

Specifies, if a reply message will be sent by *NCIQ2F* if a file has been written to disk sucessfully. It is up to a follow-up process to send another reply message at a later point in time.

Note: In the case of an error no reply message will be sent. Instead the messages belonging to the file will be moved to the dead-letter queue. If at a later time, after the messages have been moved back to the receive queue, the file will be written successfully, a reply message will be sent, if requested.

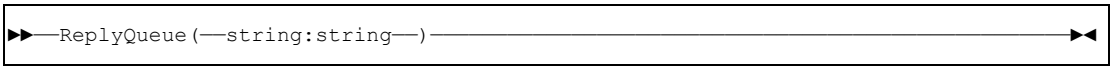


Figure 31: Reply Queue

Applicable only to NCIF2Q.

The string supplied for the ReplyQueue keyword must be a fully-qualified queue name, i.e. a queue manager name or the name of a transmission queue to a remote queue manager and a queue name separated by a colon. The queue must be local to the connect queue manager, i.e. the queue manager specified with the *PrimAddrInfo* keyword.

Note: *If a queue manager has been specified, a queue manager alias that points to a transmission queue must be defined at the sending site. If a transmission queue has been specified, a queue manager alias that resolves to the target queue manager must be defined at the receiving site.*

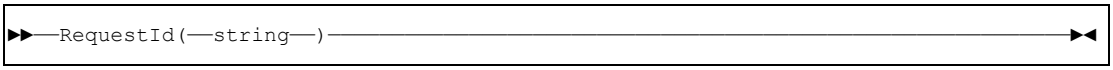


Figure 32: RequestId

Applicable only to *NCIF2Q*.

A request number, i.e. an identifying number, assigned to a file transfer request. The maximum length is 256. This parameter has no significance regarding the transmission of the file. The value specified will be contained in all meta-information messages

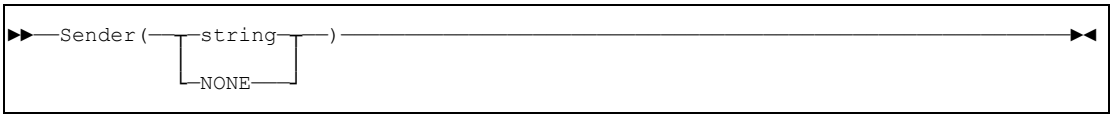


Figure 33: Sender

Applicable to *NCIF2Q* and *NCIQ2F*.

Specifies the name of the sending system. This parameter has no significance regarding the transmission of the file. The value specified will be contained in all meta-information messages

If the value **NONE** has been specified for the Sender, the following applies:

- If the symbolic section will be used as a primary symbolic section, spefifying NONE is equivalent to ommitting the Sender keyword.
- If the symbolic section will be used as a destination symbolic section, the Sender definition of the primary symbolic section will be nullified.

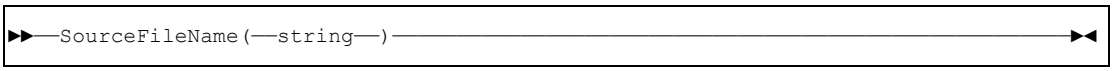


Figure 34: SourceFileName

Applicable only to *NCIF2Q*.

Specifies the name of the file to be transmitted.

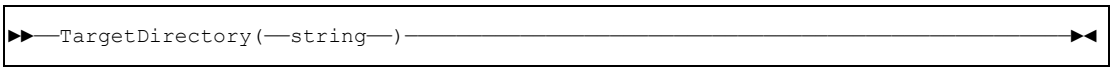


Figure 35: TargetDirectory

Applicable only to *NCIQ2F*.

Specifies the directory (i.e. high-level qualifier on z/OS) where the file at the receiving site will be written.

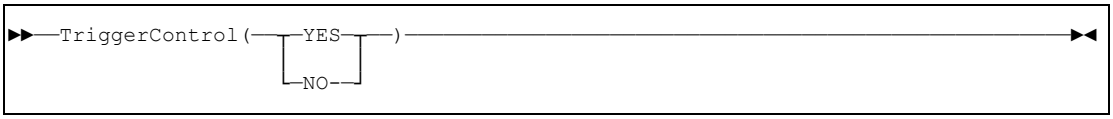


Figure 36: TriggerControl

Applicable only to *NCIQ2F*.

Specifies, if *NCIQ2F*, when started by a trigger monitor, will, in the case of unrecoverable errors try to stop triggering in order to avoid a closed loop by repetitive triggering.

Note: *Note, that this is not possible in every case. For example, if NCIQ2F is not able to access the receive queue because of a configuration error.*

After the error condition has been cleared, triggering has to be enabled manually.

YES Triggering will be disabled for unrecoverable errors, if possible

NO Triggering will never be disabled. In the case of unrecoverable errors, a closed loop will occur.

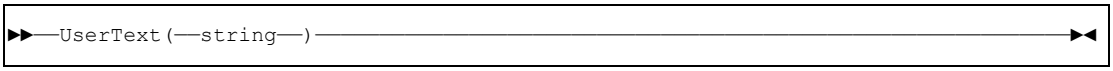


Figure 37: UserText

Applicable only to *NCIF2Q*.

Specifies a user-provided information of a maximum length of 256 byte, that will be contained in every metainformation message.

3.4 Format of meta-information and sequence of MQ messages

The sequence of MQSeries messages defining and composing a file is described in the following table

MsgId	CorrId	MsgBody	Format of MsgBody
UniqueMsgId	NCIMQFTXML.START	(empty)	None
NCIMQFTXML.1	UniqueMsgId	Block 1	String/None
NCIMQFTXML.2	UniqueMsgId	Block 2	String/None
...	...	...	...
NCIMQFTXML.N	UniqueMsgId	Block N	String/None
NCIMQFTXML.END	UniqueMsgId	Meta-information	String

- *UniqueMsgId* is the MsgId generated by the queue manager for the first message. This MsgId is repeated in the CorrId field of all the following messages belonging to the file.
- The message with the NCIMQFTXML.START string in the CorrId field serves the purpose to generate a unique MsgID value, that will be used to identify all the messages, including the meta information, belonging to a single file. The message will be deleted when the file has been processed on the receiving site.
- The message with the NCIMQFTXML.END string in the MsgId field allows the receiving program, i.e. *NCIQ2F*, to recognize when a file has arrived. The message body of the *end message* contains the the meta-information describing the file transfer request in XML format, as defined below.
- The format of the message bodies containing the file blocks is either *String* for text files of *None* for binary files.
- Note, that the MsgId and CorrId are always in ASCII format.

The meta-information is in XML format and has the following format:

```
<ncif2q>
   <nciTransportInfo>
     <type>FOLLOWUP/type>
     <destSymbolicName>Y1/destSymbolicName>
     <maxMessageSize>32K</maxMessageSize>
     <segmentCounter>8</segmentCounter>
   </nciTransportInfo>
   <fileMetaInfo>
     <logicalFileName>file1.dat</logicalFileName>
     <sourceFileName>USER1.FILE1.DAT</sourceFileName>
     <targetFileName>c:\demo\file1.dat</targetFileName>
     <fileDataType>TEXT</fileDataType>
     <fileSize>260155</fileSize>
     <maxRecordSize>81</maxRecordSize>
     <lrecl>9000</lrecl>
     <recfm>VB</recfm>
     <dsorg>ES</dsorg>
   </fileMetaInfo>
   <userData>
     <requestId>12345678</requestId>
     <replyQmgr>RQMGR1</replyQueue>
     <replyQueue>RQ1</replyQueue>
     <receiver>I050</receiver>
     <sender>TANDEM1</sender>
     <userText>any text</userText>
   </userData>
   <response>
     <returnCode>0</returnCode>
     <reasonCode>NCIF2Q_NO_ERROR</reasonCode>
     <returnMsg>File transmitted</returnMsg>
   </response>
 </ncif2q>
```

The message body of the reply and the follow-up message have the same format as the meta-information. It is the user's responsibility to evaluate these messages with suitable utilities.

4 Installation and Configuration

4.1 Installation

For installation of the *NCI MQ File Transfer Utilities* refer to the *NCI Installation and Customization Guide*.

4.2 Installation Verification

1. Install NCI (Minimum Level PNCI280SQ) into a user directory (in this example /home/SAZ1119/mqft/sun-5.6). See the *NCI Installation and Customization Guide*, chapter 1.3 Installation Sun Solaris. Perform the verification test for MQSeries.
2. Define the MQSeries objects required for the installation verification. The definitions are contained in the file *ncift.def* of the *samples* subdirectory. Perform the following command:

```
runmqsc < ncift.def
```

3. Perform the verification test for MQSeries equivalent to above with the newly defined MQSeries objects:

   Example:

```
nciping -aMQ -2 SAZ1.NCIFT.MQX1.01 -yN -cY
```

4. Define the sideinformation files. See the examples *sidef2q* and *sideq2f* in the *samples* subdirectory.
5. Start a Trigger Monitor to initiate *NCIQ2F* if a file arrives
6. Send a file with *NCIF2Q*

Don't forget to run *./setenv.sh* like in the verification test.

```
ncif2q -c sidef2q -s EBMXTEST.FILE2 -f testfile.txt
```

4.3 Configuration Example

4.3.1 Sample MQSeries definitions

for NCI File Transfer Utilities The following sample contains the commands to define the required MQSeries objects for the NCI File Transfer Utilities between Sun OS and z/OS.

```
****************************** TOP OF DATA ***************************
*
* Description:
*
*     ncift.def - Sample Definitions for NCI MQ FileTransfer Utilities
*                 used between Sun OS and OS/390 using MQSeries 5.2
*
*
* Explanation of used names:
*
*       SAZ1MQ3E stands for the Queue Manager on the Sun side
*       MQX1     stands for the Queue Manager on the OS/390 side
*
*
************************************************************************
* Definitions for receiving files via nciq2f on Sun OS
************************************************************************
*
* 1.)   Dead-letter Queue
*
*       Used for cases where receiving can't completed, even after
*       recovery...
*
define  qlocal(SAZ1.NCIFT.DEAD) +
        descr('NCIFT Dead-letter Queue') +
        defpsist(YES) +
        replace
*
* 2.)   Follow Up Queue
*
*       Used for signalling the follow up processing, after a file has
*       been received completely.
*
define  qlocal(SAZ1.NCIFT.FOLLOW.01) +
        descr('NCIFT Follow Up Queue') +
        defpsist(YES) +
        replace
*
* 3.)   Application Queue
*
*       Real application queue used for receiving files with NCIQ2F.
*
define  qlocal(SAZ1.NCIFT.RECV.01) +
        defpsist(YES) +
        initq(SAZ1.NCIFT.INITQ) trigger process(NCIFT.NCIQ2F) +
        descr('NCIFT Receiving Queue from NCIF2Q') +
        trigdata('-c /home/saz1119/mqft/sun-5.6/samples/sideq2f -s NCIQ2F.01
        trigtype(FIRST) trigmpri(1) +
        replace
*
* 4.)   Process definition
*
*       used for triggering of NCIQ2F for receiving files from
*       application queues.
*
define  process(NCIFT.NCIQ2F) appltype(UNIX) +
        applicid('/home/saz1119/mqft/sun-5.6/bin/nciq2f') +
        descr('NCIFT trigger process to start nciq2f') +
        replace
*
* 5.)   Trigger Init Queue
*
*       used for triggering of NCIQ2F
*
define  qlocal(SAZ1.NCIFT.INITQ) +
        descr('NCIFT Trigger Init Queue') +
```

```
        replace
*
* 6.)   Remote QueueManager Alias for Reply Message
*
*       For accesing remote queues with a single definition for easier
*       configuration, an alias to a remote QueueManager is used.
*
define  qremote (MQX1) +
        rqmname ('MQX1') xmitq(SAZ1.NCI.SAZ1MQ3E.MQX1.XMITQ) +
        descr('NCIFT Queue manager alias for MQX1') +
        replace
*
************************************************************************
* Definitions for sending files via ncif2q on Sun OS to an OS/390 MQSeri
************************************************************************
*
* 7.)   Remote Application Queue
*
*       Real application queue used for sending files with NCIF2Q.
*
define  qremote(SAZ1.NCIFT.MQX1.01) rqmname(MQX1) +
        rname(XB.Z41.E.NCIFT.MQX1.01) +
        xmitq(SAZ1.NCI.SAZ1MQ3E.MQX1.XMITQ) +
        descr('NCIFT Remote Queue for NCIF2Q to SYX1') +
        replace
*
* 8.)   Reply Queue
*
*       Used for for receiving the reply for the sended file.
*
define  qlocal(SAZ1.NCIFT.REPLY.01) +
        descr('NCIFT Reply Queue') +
        defpsist(YES) +
        replace

****************************** BOTTOM OF DATA ************************
```

4.3.2 Sample MQSeries Definitions for Intercommunication

The following sample contains commands to define channels, a transmission queue and a process to start a sender channel by a trigger monitor. The definitions are provided for one site of the communication path and must be defined equivalently at the other site.

Note: *The **xmitq**-parameter of the remote queue sample definition for NCIF2Q must specify the actual transmission queue name. The definitions are provide here merely for convenience. Please refer to the MQSeries documentation for details about setting up a communication infrastructure.*

```
****************************** TOP OF DATA ***************************
*
* Description:
*
*      Sample Definitions for intercommunication between two
*      MQSeries queue managers (one site only)
*
*
* Explanation of used names:
*
*       SAZ1MQ3E stands for the Queue Manager on the Sun side
*       MQX1     stands for the Queue Manager on the OS/390 side
*
*
************************************************************************
* 1.)   Transmission Queue
*
*       Used for buffering messages sent to a remote queue manager
*
define  qlocal(SAZ1.NCI.SAZ1MQ3E.MQX1.XMITQ) usage(xmitq) +
        initq(SYSTEM.CHANNEL.INITQ) trigger +
        process(SAZ1MQ3E.MQX1) +
        descr('Transmission Queue to MQX1') +
        replace
*
* 2.)   Sender Channel
*
*       Transfers messages from the transmission queue to the remote
*       queue manager
*
define  channel(SAZ1MQ3E.MQX1) chltype(sdr) trptype(tcp) +
        conname(11.22.33.44) xmitq(SAZ1.NCI.SAZ1MQ3E.MQX1.XMITQ) +
        discint(30) +
        descr('Sender Channel to MQX1) +
        replace
*
* 3.)   Receiver Channel
*
*       Transfers messages from the transmission queue to the remote
*       queue manager
*
define  channel(MQX1.SAZ1MQ3E) chltype(rcvr) trptype(tcp) +
        descr('Receiver Channel from MQX1) +
        replace
*
* 4.)   Trigger Process for Sender Channel
*
*       Process definition to a start a sender channel, when messages
*       arrive on a transmission queue
*
define  process(SAZ1MQ3E.MQX1) appltype(UNIX) +
        applicid('CSQX START') userdata(SAZ1MQ3E.MQX1) +
        descr('Process to start sender channel to MQX1') +
        replace
****************************** BOTTOM OF DATA ************************
```

4.3.3 Sample Sideinformation File for the Sending Site

The following sample contains the definitions required for NCIF2Q

```
****************************** TOP OF DATA ***************************
*
* Description:
*
*     SIDEQ2F   - Sample Definitions for NCI MQ FileTransfer Utilities
*                 used between Sun OS and OS/390 using MQSeries 5.2
*
* Explanation of used names:
*
*     SAZ1MQ3E stands for the Queue Manager on the Sun side
*     MQX1     stands for the Queue Manager on the OS/390 side
*
*********************************************************************
** Sending Site (Sun)
*********************************************************************
* General section
*
AddrType(MQ)
PrimAddrInfo(SAZ1MQ3E)                                    // Connect queue manager
ErrorMsgOpt(FILE)                                         // Error messages written to
ErrorMsgFile(/home/saz1119/mqft/sun-5.6/samples/NCIERR)  // Path and name of error
file
TraceOpt(NONE)                                            // NO error messages will be
written
TraceFile(/home/saz1119/mqft/sun-5.6/samples/NCITRC)     // Path and name of NCI
trace file
*
* Symbolic name entry for the send queues coupled with a RFTS MQ Connect
*
SymbolicName(NCIF2Q.MQX1)
 SecAddrInfo(SAZ1.NCIFT.MQX1.01)                          // remote queue name to RFTS
Connector
 SideApplData(MaxMsgSize(256K))                           // Max. size of transmitted
messages
 SideApplData(ReplyQueue(SAZ1MQ3E:SAZ1.NCIFT.REPLY.01))  // reply queue
 SideApplData(LogLevel(INFO))
 SideApplData(LogFile(/home/saz1119/mqft/sun-5.6/samples/LOGF2Q))
*
*
* Symbolic name entry for the send queues coupled with a RFTS MQ Connect
*
SymbolicName(EBMXTEST.FILE2)
 SecAddrInfo(SAZ1.NCIFT.MQX1.01)                          // remote queue name to RFTS
Connector
 SideApplData(LogicalFileName(EBMXTEST.FILE2))            // used to match RFTS TCF
entry
 SideApplData(Sender(T902SUN))                            // used to match RFTS TCF
entry
 SideApplData(MaxMsgSize(256K))                           // Max. size of transmitted
messages
 SideApplData(ReplyQueue(SAZ1MQ3E:NCIFT.REPLY.01))  // reply queue
 SideApplData(LogLevel(INFO))
 SideApplData(LogFile(/home/saz1119/mqft/sun-5.6/samples/LOGF2Q))
*
****************************** BOTTOM OF DATA ************************
```

4.3.4 Sample Sideinformation File for the Receiving Site

The following sample contains the definitions required for NCIQ2F

```
****************************** TOP OF DATA ***************************
*
* Description:
*
*       SIDEQ2F   - Sample Definitions for NCI MQ FileTransfer Utilities
*                   used between Sun OS and OS/390 using MQSeries 5.2
*
* Explanation of used names:
*
*       SAZ1MQ3E stands for the Queue Manager on the Sun side
*       MQX1     stands for the Queue Manager on the OS/390 side
*
*********************************************************************
** Receiving Site (Sun)
*********************************************************************
*General Section
*
AddrType(MQ)
PrimAddrInfo(SAZ1MQ3E)                                  // Connect queue manager name
ErrorMsgOpt(FILE)                                       // Error messages to a file
ErrorMsgFile(/home/saz1119/mqft/sun-5.6/samples/NCIERR) // Path and name of error
file
TraceOpt(NONE)                                          // NCIF2Q error messages will
be written
TraceFile(/home/saz1119/mqft/sun-5.6/samples/NCITRC)    // Path and name of NCI file
*
* Symbolic name entry for dead-letter queue
*
SymbolicName(DLQ)                                       // dead-letter-queue to store
msgs of undeliverable files
  SecAddrInfo(SAZ1.NCIFT.DEAD)
*
* Symbolic name entry for follow-up queue RFTS
*
SymbolicName(FOLLOW)                                    // Follow-up-queue to receive
meta data of finished transfers
  SecAddrInfo(SAZ1.NCIFT.FOLLOW.01)
*
* Primary symbolic name entry for receive queue in trigger data
*
SymbolicName(NCIQ2F.01)
  SecAddrInfo(SAZ1.NCIFT.RECV.01)                       // Remote queue referring to
sender
  SideApplData(TargetDirectory(/home/saz1119/mqft/sun-5.6/samples/))
  SideApplData(FollowupQueue(FOLLOW))                   // Symbolic ne of follow-up
queue
  SideApplData(RecoverCount(4))                         // # of times recovery is
attempted
  SideApplData(RecoverInterval(30))                     // Time between recovery
attempts
  SideApplData(LogLevel(TRACE))
* SideApplData(LogLevel(INFO))
  SideApplData(LogFile(/home/saz1119/mqft/sun-5.6/samples/LOGQ2F))
  SideApplData(DeadLetterQueue(DLQ))                   // Dead-letter-queue symbolic
name
*
* Example for RFTS MQ Connector TCF entry
*
SymbolicName(EBMXTEST.FILE1)                           // testcase for receiving a
sample
  SideApplData(LogicalFileName(RECVD))                 // overwrites logical file
name from sender
*
* Default for POOL transfers
*
SymbolicName(DEFAULT)
```

```
  SideApplData(FollowupQueue(NONE))                    // NO follow-up queue
*
****************************** BOTTOM OF DATA ************************
```

5 Operating Procedures and Error Handling

5.1 Operating Procedures

In the following chapters operating procedures and error situations as well as suggestions on how to deal with them are described. Error messages as well as significant events (e.g. file has been written) will be written according to the specification of the ErrorMsgOpt and ErrorMsgFile parameters specified in the sideinformation file.

If required an NCI application trace can be activated with the TraceOpt and TraceFile parameters of the sideinformation file.

If a recoverable error occurs, while *NCIQ2F* tries to re-assemble a file from MQSeries messages, retries according to definitions in the sideinformation file will be attempted.

If recovery was unsuccessful, *NCIQ2F* moves the messages belonging to a file to an application-specific dead-letter queue defined in the sideinformation file. The messages can be re-routed to the receive queue at a later time by using the MQSeries dead-letter queue handler.

Reasons for recoverable errors are for example:

- File already exists
- Messages belonging to file incomplete
- No space on volume

The messages in the dead-letter queue should be processed by the MQSeries dead-letter queue handler with an appropriate rules table, i.e. re-routed to the receive queue at a later time or discarded if recovery is not posible. If the messages will be re-routed to the receive queue and the error condition has been removed, processing continues as before.

Note: *If the messages belonging to a file will be deleted from the dead-letter queue, no reply message will be send to the sending site.*

The outcome (return and reason code) of a file transfer request are contained in the reply message put to the reply-to queue at the sending site, if such a queue has been defined in the sideinformation file.

5.2 Dealing with Error Situations when receiving a file

5.2.1 General remarks

Error situations at the receiving site are classified into two categories:

- recoverable errors
- unrecoverable errors

For **recoverable errors** *NCIF2Q* retries to write the file a given number of times according to the configuration parameters *RecoverCount* and *RecoverInterval* specified in the sideinformation file.

If the retries are not successful *NCIQ2F* tries to move the messages to the dead-letter queue specified in the sideinformation file with the *DeadLetterQueue* keyword. These messages can be moved back to the original queue by using a dead-letter queue handler or any other appropriate tool at a later time when the error situation has been resolved.

Note: *It is recommended to stop triggering while moving messages from the dead-letter queue to the receiving queue in order to avoid deadlock situations.*

There are error situations that cannot be handled in this way, for example read errors for messages on the receiving queue or write errors because the dead-letter queue is full. These are considered **unrecoverable errors**. There are two choices to deal with these errors in the case that *NCIQ2F* gets started by MQSeries triggering. They are specified through the sideinformation parameter *TriggerControl*.

TriggerControl(YES) Triggering of the receiving queue will be stopped when an unrecoverable error occurs. This means that no other files can be written until the situation has been cleared and triggering has been manually reactivated.

TriggerControl(NO) Triggering will not be stopped in case of an unrecoverable error. This means, that whenever *NCIF2Q* stops because there is no more work to do, it will get immediately started back by the trigger monitor. This could result in a loop consuming a lot of system ressources. The advantage is, that if other files arrive, they will stillc be processed.

5.2.2 Error situations and how they can be resolved

There are often different possibilities, how an error situation can be resolved, depending on the particular circumstances. It is always necessary to analyze the error situation before taking any action. In the following some of the more frequent error situations are described together with possible solutions.

File cannot be written because target file already exists

- Wait until follow-up processing has completed, which eventually deletes the target file
- Delete the existing file
- Delete the messages composing the new file from the receiving queue or the dead-letter queue

Target file cannot be created

- Grant authority to *NCIQ2F* for creating the file
- Delete the messages composing the new file from the receiving queue or the dead-letter queue and retransmit the file with adequate allocation parameters
- Edit the meta-information message associated with the file with an appropriate tool and specify adequate allocation parameters
- Change the sideinformation file parameters that lead to the error, for example path information

Sending of reply message fails or reply message doesn't show up

- Grant authority to *NCIQ2F* for putting on the reply-to queue
- Change reply-to queue parameters that could prevent putting messages to the queue like *GET(DISABLED)*, *MAXDEPTH* or *MAXMSGL*
- Define an alias for the reply-to queue queue manager at the sending site
- Correct the remote queue manager name in the queue manager alias and move the reply message from the dead-letter queue at the sending

	site to the reply-to queue Note: *If the remote queue manager was wrong in the queue manager alias at the sending site and a default transmission queue has been specified at the receiving site the reply message will be put there instead of the dead-letter queue. In this case the file has eventually be retransmitted.*
Messages required for the file are missing	• Move the messages back to the receiving queue at a later time in the hope that the missing messages have meanwhile arrived • If this doesn't help delete the messages from the dead-letter queue after an appropriate time period has elapsed and retransmit the file
Messages cannot be put to the dead-letter queue	• Define the dead-letter queue • Grant *NCIQ2F* put authority to the dead-letter queue • Change dead-letter queue parameters that could prevent putting messages to the queue like *GET(DISABLED)*, *MAXDEPTH* or *MAXMSGL* Note: *If triggering has been stopped, reactivate it after the error situation has been cleared.*
Messages will be truncated while reading them from the receiving queue	• Edit the meta-information message associated with the file with an appropriate tool and change the number of the *maxMessageSize* tag to a value that corresponds to the actual message size • Alternatively retransmit the file with an appropriate value for *maxMessageSize* Note: *This error is normally only possible, if the messages composing the file have been created by a user program. With NCIF2Q maxMessageSize will be detrmined dynamically and should be correct*
***NCIQ2F* fails to start because of errors in the sideinformation file**	• Correct the errors and restart *NCIQ2F*

Index

Backpage

T-Systems Enterprise Services GmbH
Computing Services & Solutions (CSS)
System Products & Automation (MSY-PA)
Fasanenweg 9, 70771 Leinfelden-Echterdingen, Germany

Phone : +49 89/1011-4687
Fax. : +49 711/972-91622
E-mail : cc.middleware@t-systems.com
Internet : http://www.t-systems-systemproducts.com