Middleware Suite - Application Integration

NCI - Network Computing Interface

Additional Features
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

Installation and Customization Guide for z/OS and OS/390

Version / Revision	Date	Revision
This edition relates to NCI version NCI 3.1	30/01/2006 13:06:00	165

- PNCI310/QZ05046 on z/OS, OS/390
- PNCI310/REL1003 on Unix/NT

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 Additional Features

NCI MQ File Transfer Utilities

NCI MQ Security Suite

NCI SAP R/3 RFC Server Interface

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 Tables

List of Figures

Preambel

This document describes various additional features that are available with NCI. Most of these features are built on top of the NCI base components and were introduced to provide some added values for the NCI users. Some of the features are only available for a specific platform or need a special runtime environment.

The table below shows the list of available features and their requirements. In addition, plaftform specific details are discussed in the according chapters.

Feature	Short description
NCI Gateway Service TP	The gateway service TP can pass incoming data to another target application. The gateway is able to do a protocol switch from one protocol to another protocol. The gateway is a part of the NCI Communication Manager and is available on all platforms where NCI CM is available.
NCI DB2 Service TP	Is a NCI TP that provides a DB2 SQL environment for server applications, that want to access DB2. NCI DB2 TP does the connection handling with DB2 and also the networking with the client. Note: This feature is only available for z/OS and OS/390.
NCI HTTP Gateway Servlet	Is a servlet that acts as a gateway and transforms HTTP requests to NCI requests. Note: This feature is deprecated. If you still need this feature, please contact cc.middleware@t-systems.com
NCI MDB Adapter	NCI Message Driven Beans Adapter. Note: This feature requires a J2EE 1.3 compliant Application Server.

Table 0-1: Overview - Nci Additional Features

1 NCI Gateway Service TP

The NCI Gateway Service TP is a standard NCI server application that can be used to transfer incoming data to another application. The gateway TP also supports protocol switching. That means, the incoming protocol is converted to another outgoing protocol.
Note: Only protocols that are supported by NCI, on the platform where the gateway runs, can be used.

The Gateway TP can be used, if a client application cannot directly establish a connection to the target server. Reasons for this can be for example:

- a firewall doesn't allow a direct connection.
- the server application doesn't support the client's protocol.

The typical usage of the Gateway TP is to do a protocol conversion. For example, from TCP/IP to LU 6.2, MQSeries or CICS EXCI.

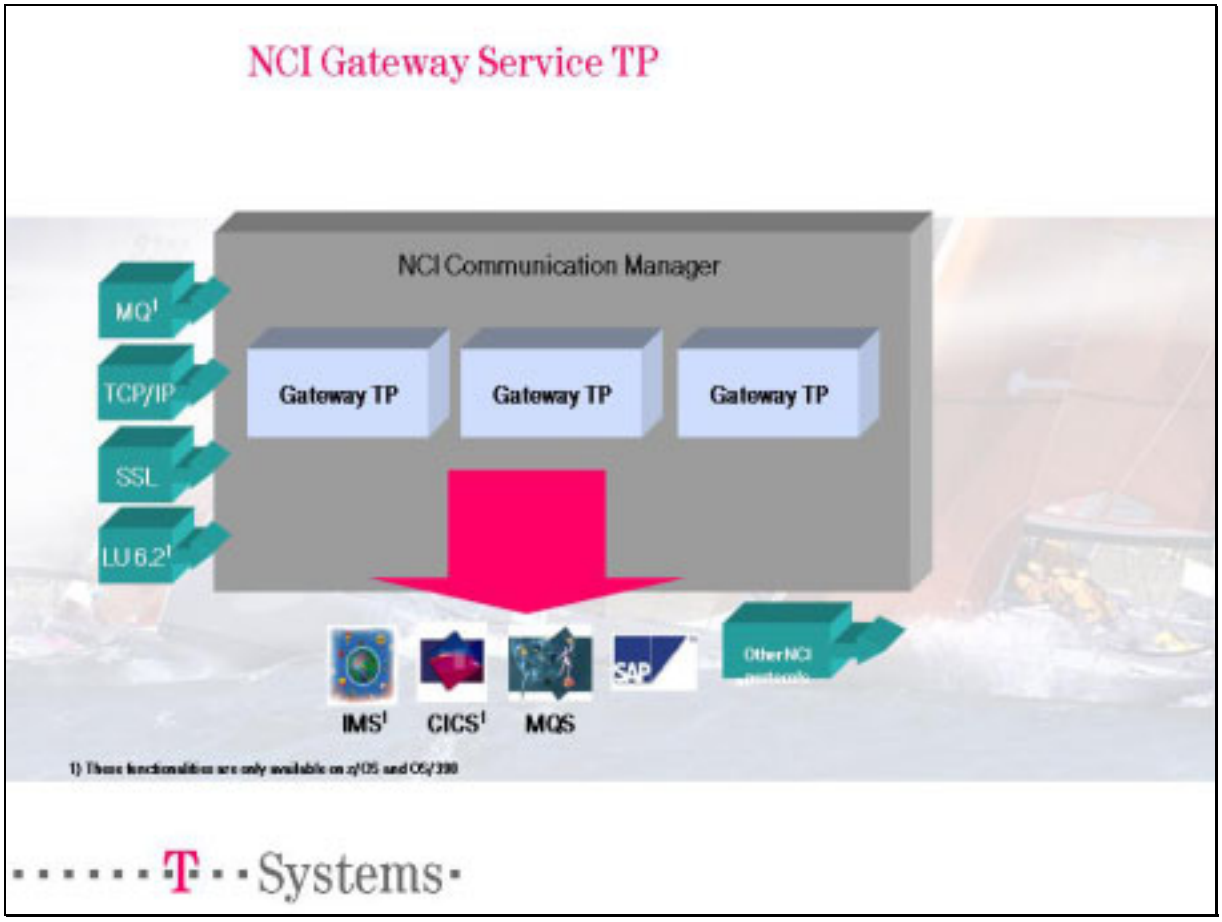


Figure 1-1: NCI Gateway Service TP

The NCI Gateway Service TP always works in a synchronous manner. That means, for the outgoing connection an NciPutGet(...) call is used. The gateway waits for the target system until the response is sent or a timeout occurs. If the gateway runs as an NCI single server, the gateway gets stuck immediately

when one outgoing connection does not respond. That's one of the reasons why we recommend to run the gateway in a NCI Communication Manager environment.

1.1 NCI Gateway Service TP – Customization and Usage

The Gateway Service TP is a standard NCI server application. It can be started as a single server application or can run under control of the NCI Communication Manager as a parallel server application. We recommend to run it within a NCI Communication Manager environment, because it offers more flexibility and load management capabilities. To setup a NCI Gateway Service TP for a NCI Communication Manager, you have to do the following steps.

1. Create a new NCI Communication Manager instance, or use an already existing instance.
   Note: One NCI Communication Manager instance can host multiple different NCI Gateway Service TPs.

2. Add a new TP definition for the Gateway Service TP to the NCI Communication Manager instance.
   To do this you have to change the NCI Communication Manager configuration. The definition of the Gateway service TP must be added to the list of TPs for this NCI Communication Manager.

3. The NCI Gateway Service TP uses the NCI sideinformation functionality to set NCI API parameters. Entries for the gateway server part and client part must be added to an existing NCI sideinformation or a new sideinformation must be created.

4. Restart the NCI Communication Manager

Note: The actions necessary, to create a new NCI Communication Manager instance, to change the configuration of an existing instance or to change/create the NCI sideinformation depend on the platform where the NCI Communication Manager is running.
Refer to *NCI Installation and Customization for Distributed Systems* or *NCI Installation and Customization for z/OS and OS/390* for detailed information.

Let's look a little bit more in detail at the TP definition for a NCI Gateway Service TP. Figure 1-2 shows a sample TP definition. The important settings are:

- TP-NAME(*name*)
  - *name*
    Can be any name of your choice. This is the name the client has to use as ServiceId to access this gateway service.

- PROGRAM-NAME(NCITPGW)
  Must be NCITPGW. This is the NCI module that implements the gateway functionality.

- PIP-DATA(*sideinfo name, symbolic name of server, symbolic name of client, buffer size, trace option*)
  Defines the parameters for the gateway itself .
  - *sideinfo name*
    Name of the NCI sideinformation file/module you use for this gateway.
    Note: NCITPGW uses NCI sideinformation to provide NCI API parameters, therefore a sideinformation file/module is needed.

- *symbolic name of server*
  Name of the section within the sideinformation that contains the parameters for the gateway server part.

- *symbolic name of client*
  Name of the section within the sideinformation that contains the parameters for the gateway client part.

- *buffer size*
  Size of the internally used buffer. The max. size of the data that is passed through the gateway must not exceed this size.

- *trace option*
  NOTRACE | Trace. Enable tracing.

```
TP-NAME(GW)
     DEFAULT-TP(NO)
     PROGRAM-NAME(NCITPGW)
     TP-MIN(0)
     TP-MAX(5)
     CYCLE-TIME(0)
     IDLE-TIME(10)
     SECURITY-LEVEL(NONE)
     TP-INSTANCE(TASK)
     PIP-DATA(NCISIDE GWServer GWClient 4096 NOTRACE)
```

Figure 1-2: Sample TP definition for NCI Gateway Service TP

In the following there are various sideinformation samples for NCI Gateway Service TPs that should illustrate the capabilities of the NCI Gateway Service TP. For a detailed description about the NCI sideinformation, refer to the *NCI Application Programming Reference* documentation.

Figure 1-3 shows a sample where the gateway passes incoming requests on to another NCI server using TCP/IP as protocoll. The following parameters specify the target server.
Protocol: TCP/IP, IP-Address: 127.0.0.1, portnumber: 4444, and TP-Name: MYAPP, timeout: 30s.
The AddrType and SecAddrInfo parameters in the GWServer section are only of interest, when the gateway runs as a single server.

```
*
* Definitions below are used for server-functions:
* Note: AddrType and SecAddrInfo are only used when the gateway runs as a single
server.
*       When running under NCI CM these parameters are ignored, because the NCI CM
TCP/IP
*       listener opens the port.
  SymbolicName(GWServer)          // Name of section
    AddrType(TCPIP)               // listen on TCP/IP
    SecAddrInfo(3333)             // use this port for listen
    ApplId(GWServer)              // set the application namem, is only used for
message proc.
*
* Definitions below are used for client-functions:
*
  SymbolicName(GWClient)
    ApplId(GWClient)              // set the application namem, is only used for
message proc.
    AddrType(TCPIP)               // addressing type: TCP/IP
    PrimAddrInfo(127.0.0.1)       // TCP/IP hostname
    SecAddrInfo(4444)             // TCP/IP portnumber
    ServiceId(MYAPP)              // TP-Name
    Timeout(30)                   // timeout after 30 seconds
```

Figure 1-3: Sideinformation definition for NCI Gateway Service TP –pass Data via TCP/IP

Figure 1-4 shows a sample sideinformation where the gateway acts as a gateway to a CICS backend. The data received from clients will be sent to a CICS Backend via the CICS EXCI protocol. The target CICS is addressed by: Name of CICS MRC PIPE: **BATCHCLI**, CICS LU-Name: **APCICS05**, CICS Transaction to be used: **TR01**. CICS program to be called: CICSPGM.
Note: This type of gateway can only run on z/OS or OS/390, because the EXCI protocol is only supported on these platforms.

```
*
* Definitions below are used for the server part of the gateway.
*
  SymbolicName(GWServer)          // Name of section
    ApplId(GWServer)              // set the application namem, is only used for
message proc.
*
* Definitions below are used for the client part of the gateway.
*
  SymbolicName(GWClient)
    ApplId(BATCHCLI)              // must match the EXCI pipe definition in the CICS
system
    AddrType(EXCI)                // addressing type: EXCI
    PrimAddrInfo(TR01)            // CICS transaction to be used
    SecAddrInfo(APCICS05)         // LU name of target CICS
    ServiceId(CICSPGM)            // Name of CICS program to be called
```

Figure 1-4: Sideinformation definition for NCI Gateway Service TP – pass data to CICS via EXCI

Figure 1-5 shows a sample sideinformation for a gateway that passes on data to MQSeries. The MQSeries connection is addressed by:
Queue manager: **MQMGR01**, Queue name: **MYAPP.QUEUE.IN**, wait a maximum of **20** seconds for a

response. Messages that were put in the queue expire after 25 seconds, if they are not read by the target application.

```
*
* Definitions below are used for the server part of the gateway.
*
  SymbolicName(GWServer)          // Name of section
    ApplId(GWServer)              // set the application namem, is only used for
message proc.
*
* Definitions below are used for the client part of the gateway.
*
  SymbolicName(GWClient)
    ApplId(MQGW)                  // Name of application
    AddrType(MQ)                  // use MQSeries
    PrimAddrInfo(MQMGR01)         // Name of queue manager
    SecAddrInfo(MYAPP.QUEUE.IN)   // Name of queue where to put in messages
    Timeout(20)                   // Timeout after 20seconds waiting for reply
    MQExpiry(25)                  // MQ Messages expires after 25seconds
```

Figure 1-5: Sideinformation definition for NCI Gateway Service TP – Pass data to MQSeries

2 NCI DB2 Service TP

The NCI DB2 Service TP provides a DB2 SQL environment for NCI Server applications. The NCI DB2 Service TP handles all the DB2 connection stuff. Connections can be made using the RRSAF or CAF DB2 attachment interfaces.
Furthermore, the NCI DB2 Service TP handles all the NCI networking issues that usually have to be done in the application code itself. The application code is called by the NCI DB2 Service TP code using a standardized interface. The application program must be written according to this interface.

Note: The NCI DB2 Service TP is only available for NCI on z/OS and OS/390.

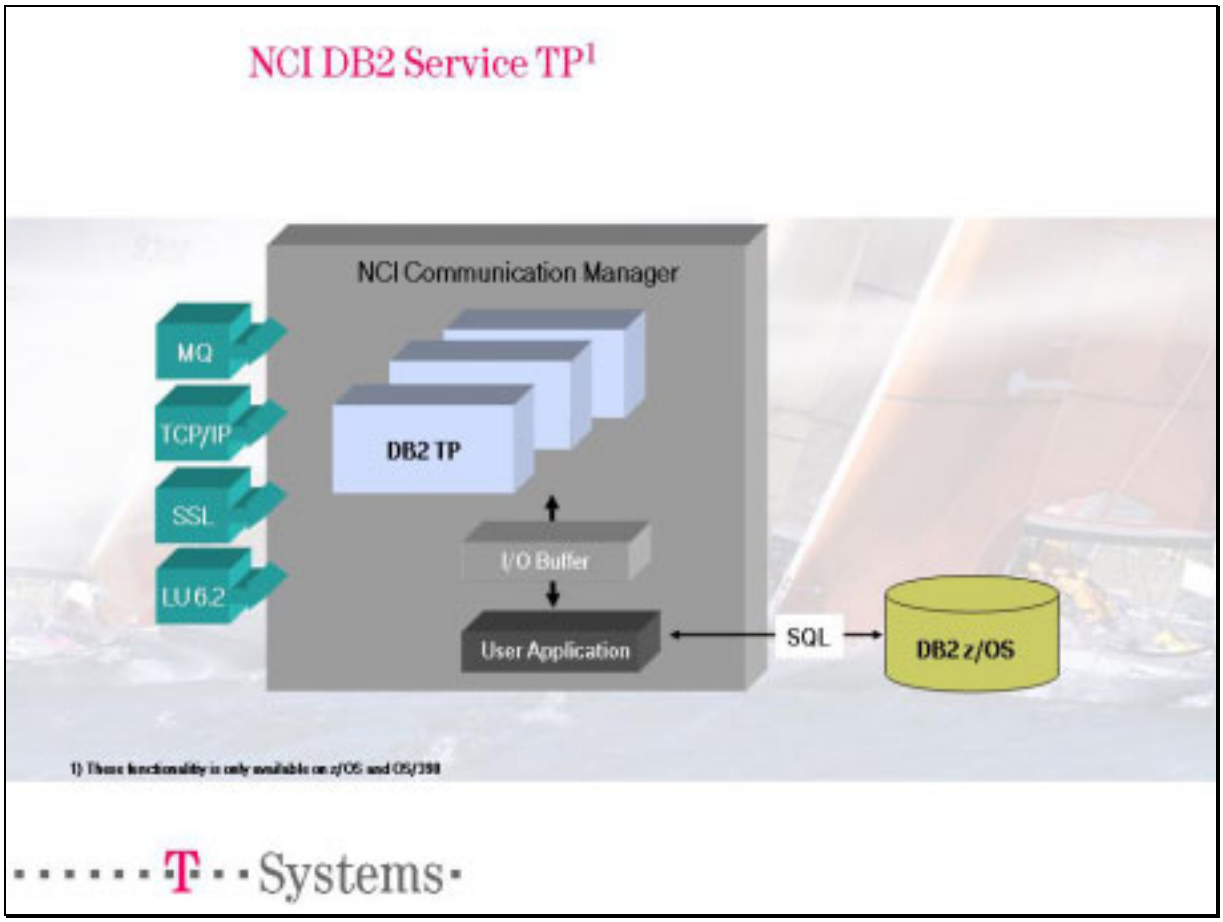


Figure 2-1: NCI DB2 Service TP – Overview

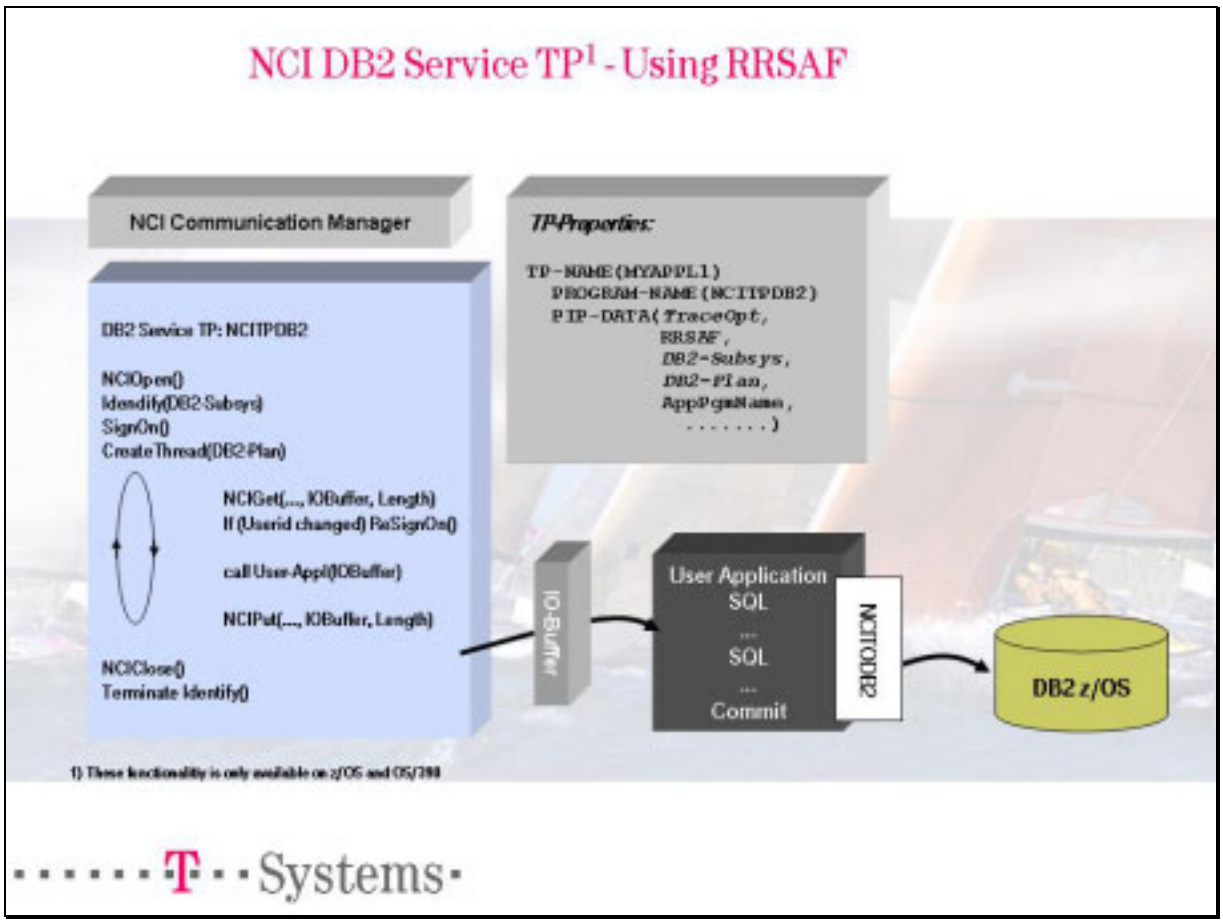


Figure 2-2: NCI DB2 Service TP - Using RRSAF

Figure 2-1and Figure 2-2 illustrate how the NCI DB2 Service TP generally works. If a client request is received by the NCI DB2 Service TP, it is first checked, if a connection to the DB2 subsystem already exists. If not, a new connection will be established. If a connection already exists and the userid of the client is the same as the previous one, the connection will be reused. If the userid of the client changed, a new *SIGNON* to DB2 will be done with the security environment of the new client. If CAF is used, the connection to DB2 will be terminated and reestablished each time, because the CAF interface does not support a resigon.

2.1 NCI DB2 Service TP – Customization and Usage

The NCI DB2 Service TP must be started under control of the NCI Communication Manager for z/OS and OS/390.
We recommend to set the TP-attribute *SECURITY-LEVEL(PROPAGATE)* to make sure, that the TP instance (MVS TCB) will run under the security environment of the requesting client client. In this case the userid of the client will be passed to the DB2 subsystem. Authorization checks within DB2 will then be done with the client's userid.

Configuration parameters needed by the NCI DB2 Service Interface must be setup in the NCI

configuration module. A TP entry for the NCI DB2 Service TP must be added to the NCI Communication Manager configuration module (see Figure 2-3). The NCI DB2 Service TP specific parameters must be specified with the PIP-DATA parameter.

The following positional parameters are supported:

1. Trace options (TRACE | NOTRACE)
2. DB2 interface to use (RRS | CAF)
3. DB2 subsystem name (1-4 characters)
4. DB2 Plan (1-8 characters)
5. Maximum I/O buffer size (numeric value 1-99999999)
6. Name of user application program to be called (1-8 characters)
   There are 3 possibilities to define the user application program
   - PGMName
     User application program is called as a subroutine. Working Storage in Cobol modules is only initialized during the first invocation
   - (PGMName,SUBR)
     User application program is called as a subroutine. Working Storage in Cobol modules is only initialized during the first invocation
   - (PGMName,MAIN)
     User application program is called as a main routine. Working Storage in Cobol modules is initialized for each invocation
7. Storage allocation (ABOVE | BELOW)
8. Timeout (Timeout in seconds. This timeout is used while receiving data from the client)
9. (currently unused)
10. Parameters passed to user application (1-100 characters)

```
* DB2 interface used:              RRSAF
* DB2 subsystem ID:                DB2C
* Name of Plan:                    PGM1PL
* Size of IO-Buffer:               32777 bytes
* Name of user appl. program:      PGM1 (called as subroutine)
* IO-Buffer allocation:            Below
*
   TP-NAME(MYAPP1)
     DEFAULT-TP(NO)
     PROGRAM-NAME(NCITPDB2)
     TP-MIN(0)
     TP-MAX(5)
     IDLE-TIME(10)
     TP-INSTANCE(TASK)
     SECURITY-LEVEL(PROPAGATE)
     PIP-DATA(NOTRACE,RRS,DB2C,PGM1PL,32777,PGM1,B,,,USERPRMS)
*
* DB2 interface used:              RRSAF
* DB2 subsystem ID:                DB2C
* Name of Plan:                    PGM2PL
* Size of IO-Buffer:               32777 bytes
* Name of user appl. program:      PGM2 (called as main)
* IO-Buffer allocation:            Above
* Timeout:                         10 seconds
*
   TP-NAME(MYAPP2)
     DEFAULT-TP(NO)
     PROGRAM-NAME(NCITPDB2)
     TP-MIN(0)
     TP-MAX(5)
     IDLE-TIME(10)
     TP-INSTANCE(TASK)
     SECURITY-LEVEL(PROPAGATE)
     PIP-DATA(NOTRACE,RRS,DB2C,PGM2PL,32777,(PGM2,MAIN),A,10)
```

Figure 2-3: Sample TP definitions for the NCI DB2 Service TP

Data received from the client will be passed to the user application via the following interface. The user application must be written according this interface!

1. User application parms (10th parameter of *PIP-DATA*), (1-100 characters, Input)
2. Length of input data (data received from client), (Integer, Input)
3. Maximum length of I/O buffer, (Integer, Input)
4. Length of output data (data to be sent to the client) (Integer, Output)
5. I/O buffer (at input: data received from client, at output: data to be sent to the client), (Variable, In/Output)
6. Userid of the client, (1-8 characters, Input)
7. Return Code (must be set by user application), (Integer, Output)
   Meaning of Return Codes:
   - **0** - OK, continue processing
   - **4** - Error, close DB2 connection and continue
   - **8** - Error, terminate server TP

The logic of the user application should be as follow:

1. Verify input parameters (I/O-buffer, userid, ...)
2. Perform SQL statements
3. Setup I/O buffer (data to be sent to the client)
4. Setup length of output data (Length of data to be sent to the client)

5. Setup Return Code

Important Notes:

1. If the user application is written in Cobol the Compiler option: *NODYNAM* must be used.
2. If the user application is written in Cobol the program called from the NCI DB2 Service TP must use GOBACK to return
3. You cannot use different attachment types (RRSAF,CAF) in the same address space.
4. The user application must be link-edited with the **NCITODB2** stub, to support dynamic selection of the DB2 interface (RRSAF or CAF). For detailed description refer to chapter 2.1.1-NCI DB2 Stub

The next two figures show a cobol example that is written according to the NCI DB2 Service TP interface.

```
      ****************************************************************
      *  NCI2DB2                                                     *
      *                                                              *
      *  CALLED BY :  NCI SERVICE TP: NCITPDB2                       *
      *                                                              *
      *  ATTRIBUTES : REENTERABLE                                    *
      *               AMODE=31                                       *
      *               RMODE=ANY                                      *
      *                                                              *
      *  INTERFACE FOR INPUT:                                        *
      *                                                              *
      *    - USER APPL PARMS                 (INPUT/CHAR 100)        *
      *    - LENGTH OF INPUT DATA            (INPUT/INT)             *
      *    - MAXIMUM LENGTH OF IO-BUFFER     (INPUT/INT)             *
      *    - LENGTH OF OUTPUT DATA           (OUTPUT/INT)            *
      *    - IO-BUFFER                       (IN-OUTPUT/CHAR VARIABLE)*
      *    - CLIENT'S USERID                 (INPUT/CHAR 8)          *
      *    - RETURN CODE                     (OUTPUT/INT)            *
      *      - 0: OK, CONTINUE                                       *
      *      - 4: ERROR, CLOSE DB2 CONNECTION                        *
      *      - 8: ERROR, TERMINATE SERVER TP                         *
      ****************************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. NCI2DB2.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
      *
       77 W-RETURN-CODE               PIC ++++9.
      *
            EXEC SQL INCLUDE SQLCA END-EXEC.
            EXEC SQL INCLUDE EMP END-EXEC.
      *
       LINKAGE SECTION.
      *
       77  APPL-PARMS                 PIC X(100).
       77  APPL-INPUT-LENGTH          PIC S9(9) COMP.
       77  APPL-MAX-BUFFSIZE          PIC S9(9) COMP.
       77  APPL-OUTPUT-LENGTH         PIC S9(9) COMP.
      * APPL-IOBUFFER can be of any length
       01  APPL-IOBUFFER.
           05 FILLER                  PIC X(32768).
      *
       01  APPL-IOBUFFER-TAB REDEFINES APPL-IOBUFFER.
           05  W-MSGTEXT OCCURS 1057 INDEXED BY I-IOBUFFER.
               10 W-TEXT1                  PIC X(7).
               10 W-EMPNO                  PIC X(6).
               10 W-TEXT2                  PIC X(6).
               10 W-FIRSTNME               PIC X(12).
      *
       77  APPL-USERID                PIC X(8).
       77  APPL-RETURN-CODE           PIC S9(9) COMP.
      *
       PROCEDURE DIVISION USING APPL-PARMS        APPL-INPUT-LENGTH
                                APPL-MAX-BUFFSIZE APPL-OUTPUT-LENGTH
                                APPL-IOBUFFER     APPL-USERID
                                       APPL-RETURN-CODE.
```

Figure 2-4: Sample Cobol Application called by the NCI DB2 Service TP – Part 1 of 2

```
 S00 SECTION.
*
     EXEC SQL
          DECLARE CURS1 CURSOR FOR
          SELECT EMPNO, FIRSTNME
          FROM DSN8510.EMP
     END-EXEC
*
     EXEC SQL
          OPEN CURS1
     END-EXEC
     IF SQLCODE NOT = 0 THEN
       MOVE 4 TO APPL-RETURN-CODE
       MOVE SQLCODE TO W-RETURN-CODE
       DISPLAY "NCI2DB2 - ERROR ON OPEN CURSOR, SQLCODE = "
               W-RETURN-CODE UPON SYSOUT
       GO TO S99
     END-IF
*
     SET I-IOBUFFER TO 1
     MOVE 0 TO APPL-OUTPUT-LENGTH
*
     PERFORM WITH TEST BEFORE                                        7
             UNTIL APPL-MAX-BUFFSIZE < LENGTH OF W-MSGTEXT
       EXEC SQL
            FETCH CURS1
            INTO :EMPNO, :FIRSTNME
       END-EXEC
       IF SQLCODE NOT = 0 THEN GO TO S90 END-IF
*
       MOVE EMPNO     TO W-EMPNO (I-IOBUFFER)
       MOVE SPACE     TO W-FIRSTNME (I-IOBUFFER)
       MOVE FIRSTNME-TEXT (1:FIRSTNME-LEN)
                      TO W-FIRSTNME (I-IOBUFFER)
       MOVE "EMPNO: " TO W-TEXT1 (I-IOBUFFER)
       MOVE " FNM: " TO W-TEXT2 (I-IOBUFFER)
       SET I-IOBUFFER UP BY 1
*
       ADD LENGTH OF W-MSGTEXT TO APPL-OUTPUT-LENGTH
     END-PERFORM
     .
*
 S90.
     EXEC SQL
          CLOSE CURS1
     END-EXEC
*
     EXEC SQL
          COMMIT
     END-EXEC
     .
 S99.
     GOBACK.
```

Figure 2-5: Sample Cobol Application called by the NCI DB2 Service TP – Part 2 of 2

2.1.1 NCI DB2 Stub

DB2 provides three different interfaces depending on the program execution environment.

- DSN (if running under TSO environment)

- CAF (Call Attachement Facility)
- RRSAF (RRS Attachement Facility)

All interfaces have the same SQL API. The application developer has to choose one of these interfaces during program linkedit phase, by including one of the following DB2 stubs.

- INCLUDE OBJ(DSNELI) ===> TSO DSN
- INCLUDE OBJ(DSNALI) ===> CAF
- INCLUDE OBJ(DSNRLI) ===> RRSAF

All interfaces provide the **DSNHLI** entry, which will be used for all SQL calls.

If a program wants to select the DB2 interface dynamically at runtime, the NCI DB2 stub (**NCITODB2**) must be included during linkedit instead of a DB2 stub. **NCITODB2** loads the desired DB2 stub dynamically during runtime and then passes control to the DB2 code. The DB2 stub (DSNELI, DSNALI or DSNRLI), that will be used is determined as follow:

1. DB2 interface name already set by a higher level task (e.g. NCI DB2 Service TP). The name of the interface (RRS | CAF) will be passed to the NCI DB2 stub via z/OS aor OS/390 name token services.
2. DB2 interface name is set via parameter passed to entry point: NCITODB2 (open-entry)
3. DB2 interface name will be determined by itself as follow.
   - If TSO environment exists: DSN will be used
   - If TSO environment does not exist: RRSAF will be used

```
//COBOL    EXEC PGM=IGYCRCTL,REGION=4096K,
// PARM=(RENT,RESIDENT,OPTIMIZE,NODYNAM)
....
//LKED     EXEC PGM=IEWL,PARM='RENT,XREF,LET,LIST,MAP'
//SYSPRINT DD   SYSOUT=*
//OBJ      DD   DSN=SYS4.LINKLIB,DISP=SHR
//SYSLIB   DD   DSN=SYS1.SCEELKED,DISP=SHR
//         DD   DSN=ISSDB.V26.DB2510.SDSNLOAD,DISP=SHR
//SYSLMOD  DD   DSN=SAZ1.USER.LOADLIB(USERPROG),DISP=SHR
//SYSUT1   DD   UNIT=SYSDA,DISP=(,DELETE),SPACE=(CYL,(10,1),RLSE)
//SYSLIN   DD   DSN=&&OBJDSET,DISP=(OLD,DELETE)
//         DD   DDNAME=SYSIN
//SYSIN    DD   *
  INCLUDE OBJ(NCITODB2)
  ENTRY USERPROG
  NAME USERPROG(R)
```

Figure 2-6: Sample Compile and Link edit Job for Cobol using the NCI DB2 Stub

3 NCI Java Servlets

A servlet is a Java programming language class used to extend the capabilities of servers that host applications accessed via a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers.

List of available NCI Java Servlets:

Servlet name	Description
NciHttpGateway	Gateway to pass HTTP requests to other applications. Note: This feature is deprecated. If you still need this feature please contact cc.middleware@t-systems.com

3.1 NCI HTTP Gateway Servlet

Note: This feature is deprecated.
If you still need this feature please contact cc.middleware@t-systems.com

NciHttpGateway is a Java Servlet and allows to forward HTTP requests to other applications using the NCI Application Programming Interface (API). The HTTP gateway function is useful for client applications that cannot directly establish a connection to a server (e.g. firewall doesn't allow a direct connection or the server application doesn't support the clients protocol). NciHttpGateway uses the NCI Pure Java Interface and therefore supports the protocols TCP/IP, HTTP and MQSeries.

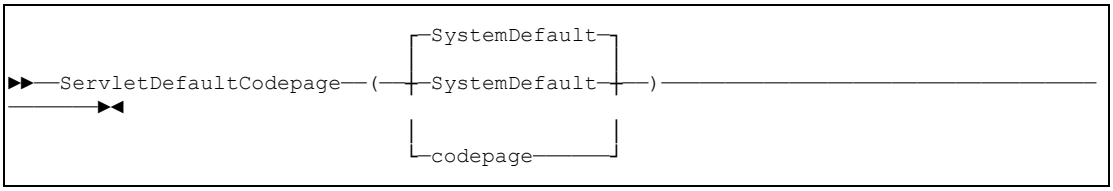
The NciHttpGateway servlet can run within any application server that has an appropriate servlet engine.

3.1.1 NciHttpGateway Initialization Parameters

The NciHttpGateway accepts the following servlet initialization parameters that are processed by the servlet itself.

ServletDefaultCodepage

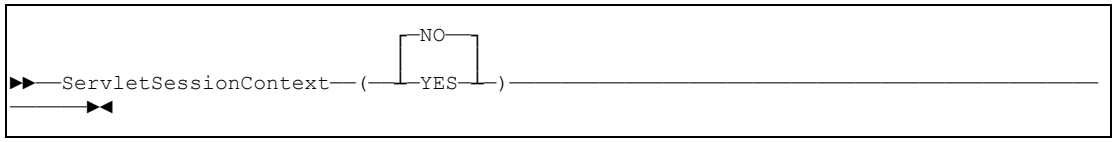
To interpret the HTTP Basic Authorization Header, NciHttpServlet uses the codepage of the HTTP client. If this codepage is not available on server (signaled by java.io.UnsupportedEncodingException), the ASCII codepage *ISO8859_1* will be used to read the authentication information. The initialization parameter *ServletDefaultCodepage* allows to set another default codepage or to use the system default codepage of the server.



SystemDefault The system default codepage is being used.
codepage Any other valid codepage.

ServletSessionContext

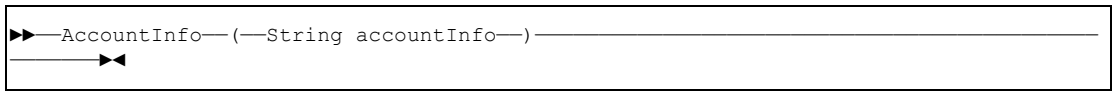
For each HTTP request, the servlet service method will be dispatched as a separate thread. Therefore each servlet instance needs to construct a new Nci object. For performance reasons, the Nci object reference can be stored via servlet session context. For subsequent HTTP requests from the same client, the Nci object can be reused.
Note: It is not possible to store the Nci session context to a persistent data store. For more information refer to the specific documentation of your application server.

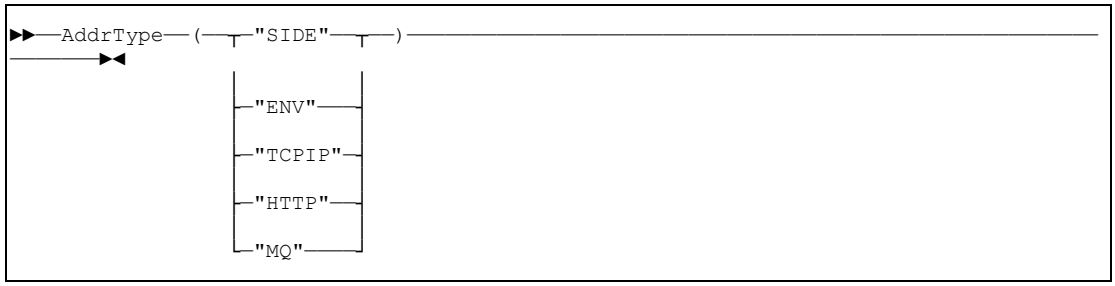


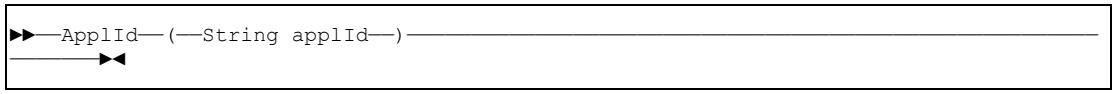
YES	The Nci object reference is stored via servlet session context for later reuse.
NO	The Nci object reference is released after each HTTP request.

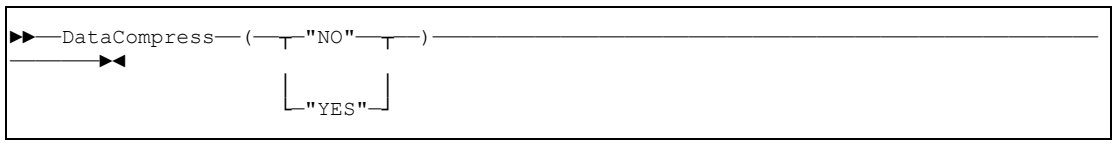
NciHttpGateway NCI API specific initialization parameters

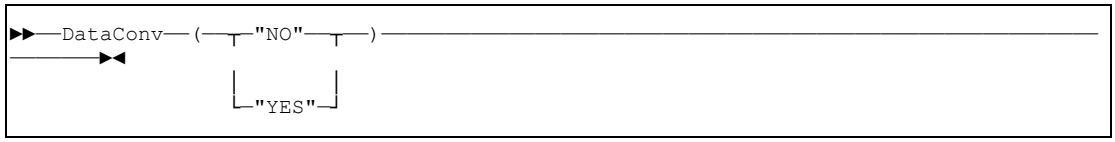
NciHttpGateway accepts the following servlet initialization parameters that are passed to the NCI API. For a complete description of those parameters refer to the *NCI Application Programming Reference* documentation.

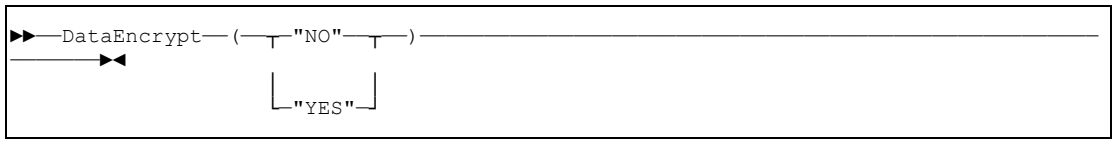


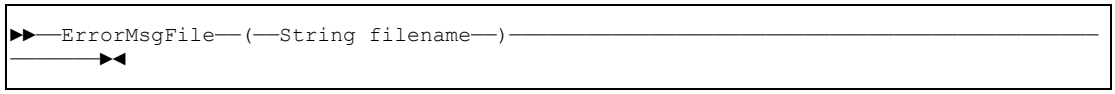


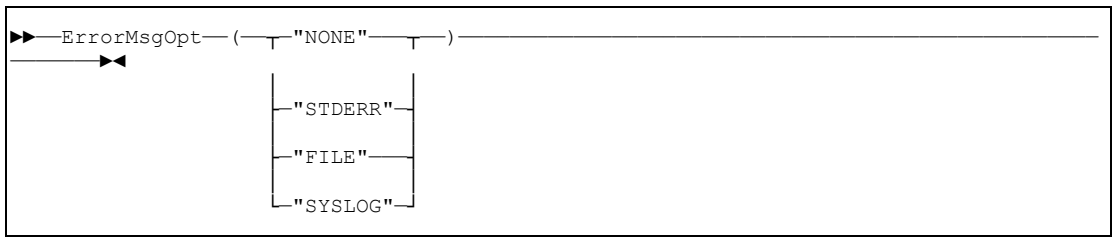


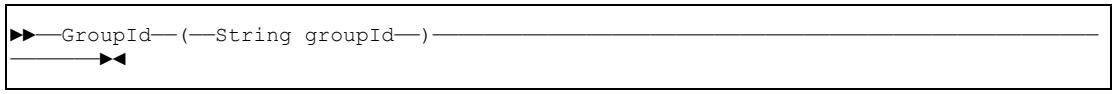


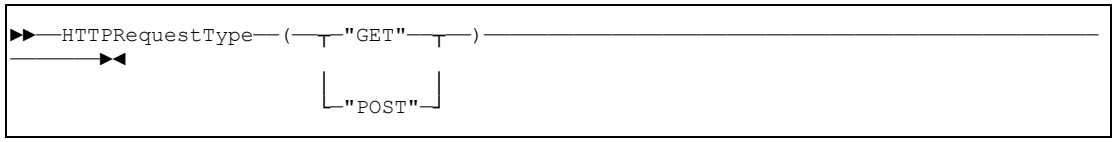


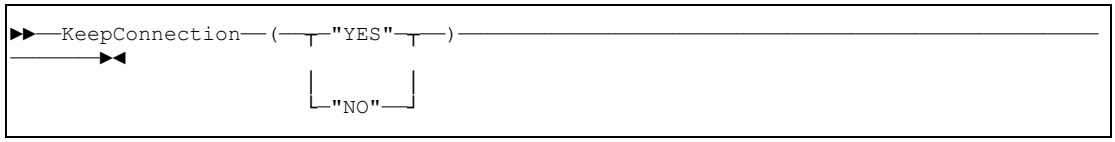

DataEncrypt
"NO"
"YES"

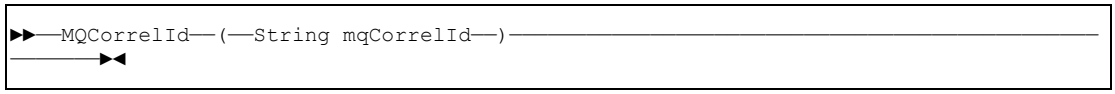

ErrorMsgFile
String filename

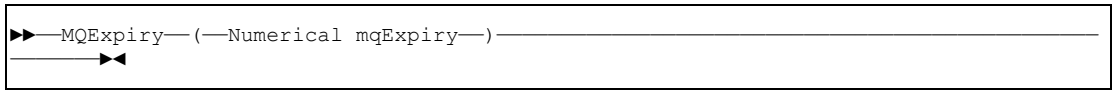

ErrorMsgOpt
"NONE"
"STDERR"
"FILE"
"SYSLOG"

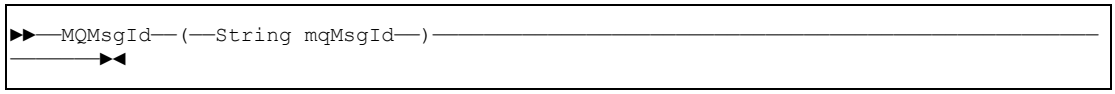

GroupId
String groupId

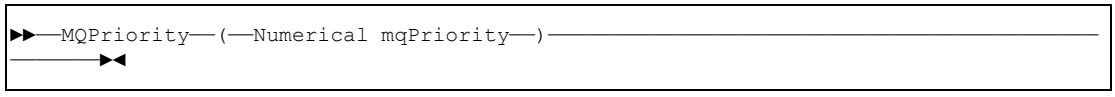

HTTPRequestType
"GET"
"POST"

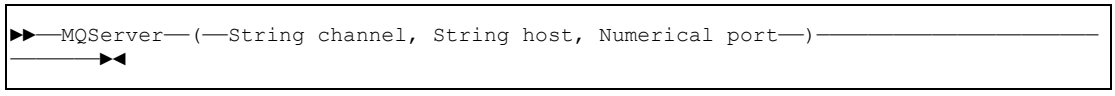

KeepConnection
"YES"
"NO"

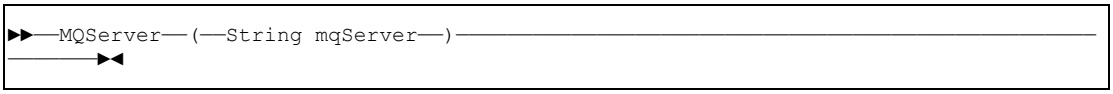

MQCorrelId
String mqCorrelId

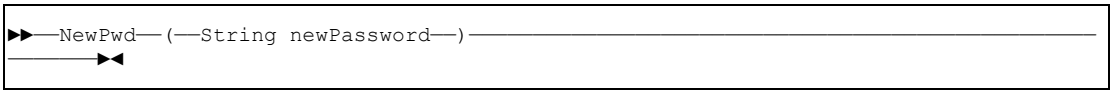

MQExpiry
Numerical mqExpiry

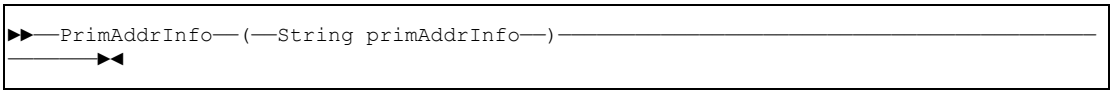

MQMsgId
String mqMsgId

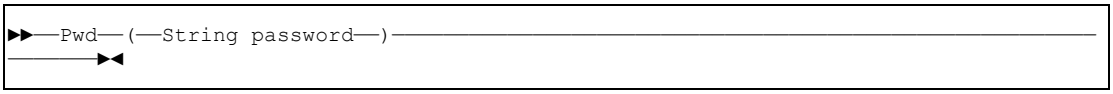

MQPriority
Numerical mqPriority

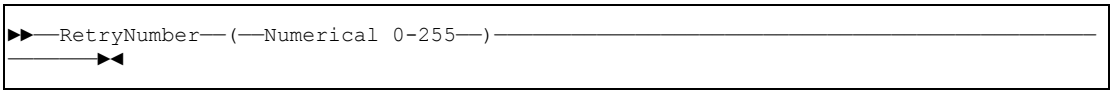

MQServer
String channel, String host, Numerical port

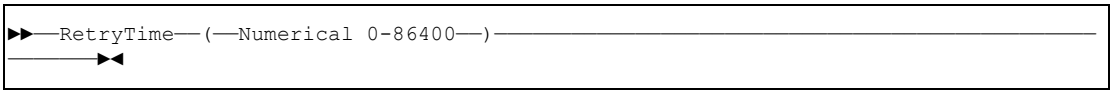

MQServer
(
String mqServer
)

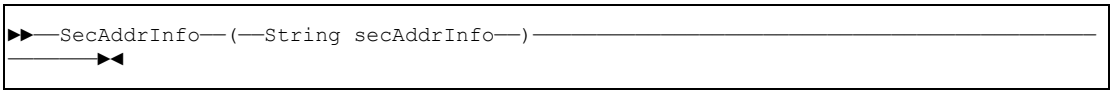

NewPwd
(
String newPassword
)

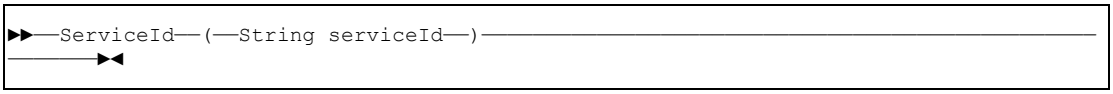

PrimAddrInfo
(
String primAddrInfo
)

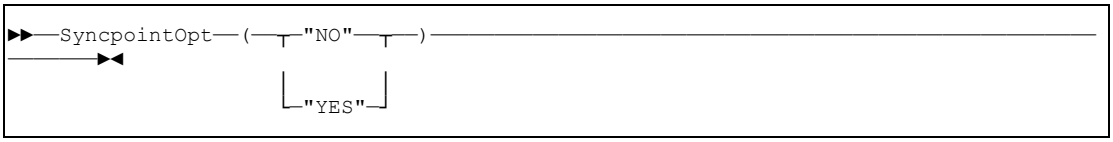

Pwd
(
String password
)

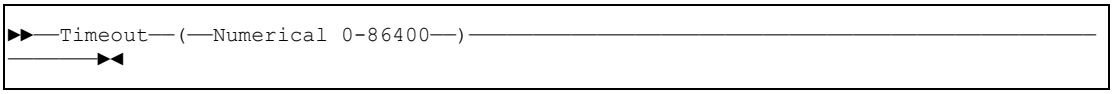

RetryNumber
(
Numerical 0-255
)

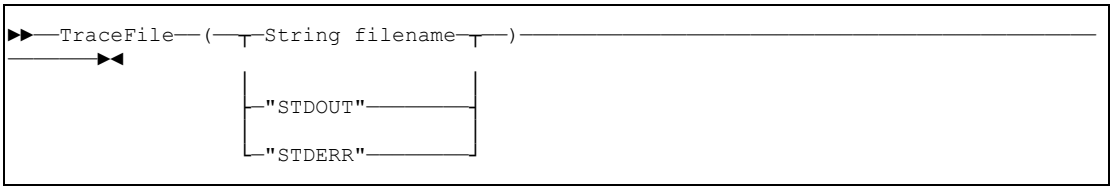

RetryTime
(
Numerical 0-86400
)

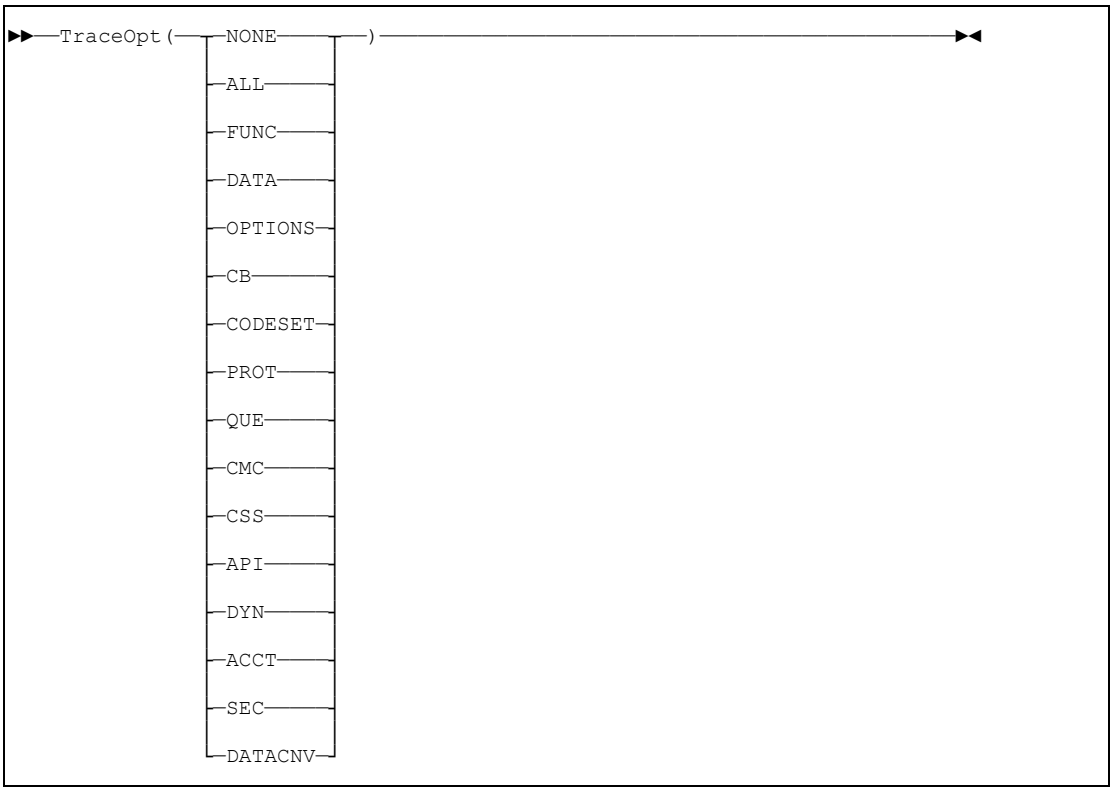

SecAddrInfo
(
String secAddrInfo
)

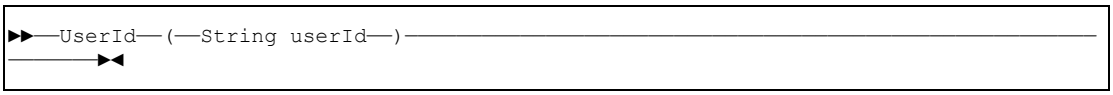

ServiceId
(
String serviceId
)


SyncpointOpt
(
"NO"
"YES"
)


Timeout
(
Numerical 0-86400
)


TraceFile
(
String filename
"STDOUT"
"STDERR"
)





Sample Web Server configurations for NciHTTPGateway

```
<?xml version="1.0"?>
<!DOCTYPE was-config "$XMLConfigDTDLocation$$dsep$xmlconfig.dtd">
<was-config>
  <virtual-host name="wassxt" action="update">
    <alias-list>
      <alias>www.sxt.com</alias>
      <alias>r30ntest.sxt.com</alias>
      <alias>r30ntest.wassxt.com</alias>
    </alias-list>
    <uri name="/SxtApp1/NCIGW" rootURI="/SxtApp1" action="create"/>
    <uri name="/SxtApp1/DIAWEBT" rootURI="/SxtApp1" action="create"/>

  </virtual-host>
          <classpath>
            <path value="c:\www\common\servlets\nci\nci.jar"/>
            <path value="c:\www\common\servlets\nci\nciservlets.jar"/>
          </classpath>

          <servlet name="NCIGW" action="update">
            <description>NCI Http Gateway</description>
            <code>com.tsystems.nci.servlets.NciHttpGateway</code>
            <init-parameters>
              <parameter name="ApplId" value="NCIGW"/>
              <parameter name="AddrType" value="TCPIP"/>
              <parameter name="PrimAddrInfo" value="53.113.127.10"/>
              <parameter name="SecAddrInfo" value="14444"/>
              <parameter name="ServiceId" value="PWDCHECK"/>
              <parameter name="Timeout" value="10"/>
              <parameter name="TraceFile" value="C:\tmp\ncitrace.log"/>
              <parameter name="ServletSessionContext" value="Y"/>
              <parameter name="TraceOpt" value="ALL"/>
              <parameter name="DataConv" value="Y"/>
            </init-parameters>
            <load-at-startup>false</load-at-startup>
            <debug-mode>false</debug-mode>
            <uri-paths>
              <uri value="/NCIGW"/>
            </uri-paths>
            <enabled>true</enabled>
          </servlet>
```

Figure 3-1: IBM WebShere Application Server 3.5 for Windows NT

Example to access NciHttpGateway via Web Browser

```
http://www.sxt.com/SxtApp1/NCIGW?HelloWorld
```

Example to access NciHttpGateway via Java application *Nciping*

```
java -classpath ..\bin\nciping.jar;..\bin\nci.jar; com.tsystems.nci.tools.NciPing
     -a HTTP -1 www.sxt.com -2 80 -3 SxtApp1/NCIGW -u DB2PUBL -p PWD??
```

```
appserver.classpath=/nci/java/bin/nci.jar:/nci/java/bin/nciservlets.jar

webapp.SxtApp1.servlet.NCIGW.servletmapping=/NCIGW
webapp.SxtApp1.servlet.NCIGW.code=com.tsystems.nci.servlets.NciHttpGateway
webapp.SxtApp1.servlet.NCIGW.initargs=
  ApplId=NCIGW,
  AddrType=TCPIP,
  PrimAddrInfo=127.0.0.1,
  SecAddrInfo=14444,
  ServiceId=PWDCHECK,
  TraceOpt=ALL,
  TraceFile=STDOUT,
  DataConv=Y,
  Timeout=30,
  ServletSessionContext=N,
  ServletDefaultCodepage=ISO8859_1
```

Figure 3-2: IBM Webshere Application Server 3.5 for OS/390

Example to access NciHttpGateway via Web Browser

```
http://53.113.127.10:9912/SxtApp1/NCIGW?HelloWorld
```

Example to access NciHttpGateway via Java application *Nciping*

```
java -classpath ..\bin\nciping.jar;..\bin\nci.jar; com.tsystems.nci.tools.NciPing
     -a HTTP -1 53.113.127.10 -2 9912 -3 SxtApp1/NCIGW -u DB2PUBL -p PWD??
```

Programming sample to access NciHttpGateway

```
import com.debis.nci. *;

public class NciHttp {

public NciHttp() {
 super();
}

public static void main(java.lang.StringÝ¨ args) {

int rc; // Return code variable

NciMessage outMessage = new NciMessage("Hello World"); // msg to be sent
NciMessage inMessage; = new NciMessage();          // received message

Nci nci = new Nci(); // Create an instance of NCI

try
 {
       nci.setAddrType("HTTP");      // Set AddressingType: HTTP
       if (args.length >= 1)
              nci.setPrimAddrInfo(argsÝ0¨); // Set TCP/IP hostname or IP address
       if (args.length >= 2)
              nci.setSecAddrInfo(argsÝ1¨); // Set TCP/IP portnumber
       nci.setServiceId("SxtApp1/NCIGW"); // Set host file-name (Servlet path)
       nci.setApplId("NciHttp"); // Set Application Identifier

       System.out.println("Message being sent: " + outMessage.getMessage() +
                          "  (" + outMessage.getLength() + ")" );

       nci.putGet(outMessage,inMessage); // Put a msg and wait for a reply   msg

       System.out.println("Received message: " + inMessage.getMessage() +
                                    "  (" + inMessage.getLength() + ")" );
 }

catch (IllegalArgumentException e)
 {
       System.out.println(e.getMessage());
 }
catch (NciInfoException e)
 {
              int errorResonCode = nci.getErrorReasonCode();
              // Analyze error ResonCode and ro recovery (e.g. promt user for pwd,
...)
              System.out.println("Info: " + nci.getErrorMsgText() );
 }
catch (NciException e)
 {
              System.out.println("Error: " + nci.getErrorMsgText() );
 }
finally
       {
              nci.close();
       }
}
}
```

Figure 3-3: Example Nci HTTP client application

4 NCI Enterprise Java Beans

In the words of Sun Microsystems: Enterprise JavaBeans is a component architecture for the development and deployment of component-based business applications. The EJB server-side model simplifies the development of middleware applications by providing automatic support for services such as transactions, security, and database connectivity.

List of available NCI Enterprise Java Beans:

Bean name	Description
NciMDBAdapter	Message Driven Bean, offering nciPutGet capability for Enterprise Applications using JMS
NciMDBPong	Sample Bean, implementing the NciMsgHandler interface. Used as sample application bean called by NciMDBAdapter.

4.1 NCI MDB Adapter

A Message Driven Bean (MDB) is a specific form of an Enterprise Java Bean, which implements the interfaces MessageDrivenBean and MessageListener. It is invoked by a Port-Listener, which has to be defined on the application server hosting the MDB, if a message arrives on a JMS queue. The MDB is invoked with the JMS message as parameter. Because it runs in a transaction, messages are not lost, but are available for retry, if an error occurs during the processing.
The application logic, used to process the message and produce a response has to be implemented as a separate EJB implementing the interfaces NciMDBHandler and NciMDBHandlerHome.

The NciMDBAdapter can run within a WebSphere Application Server on any operating system. There may also be a dependency to IBM WebSphere MQ as Messaging Middleware, because of WebSphere MQ specific JMS message header properties.

4.1.1 NciMDBAdapter Installation and Customization

Before deploying the NciMDBAdapter following prerequisites have to be satisfied on the application server.

QueueConnectionFactory

Required as connect queue manager for the Port-Listener and the NciMDBAdapter to send the reply message. They can be identical, but don't have to.

Sample for a QueueConnectionFactory:

```
Resources > WebSphere MQ JMS Provider > WebSphere MQ Queue Connection Factories > New
```

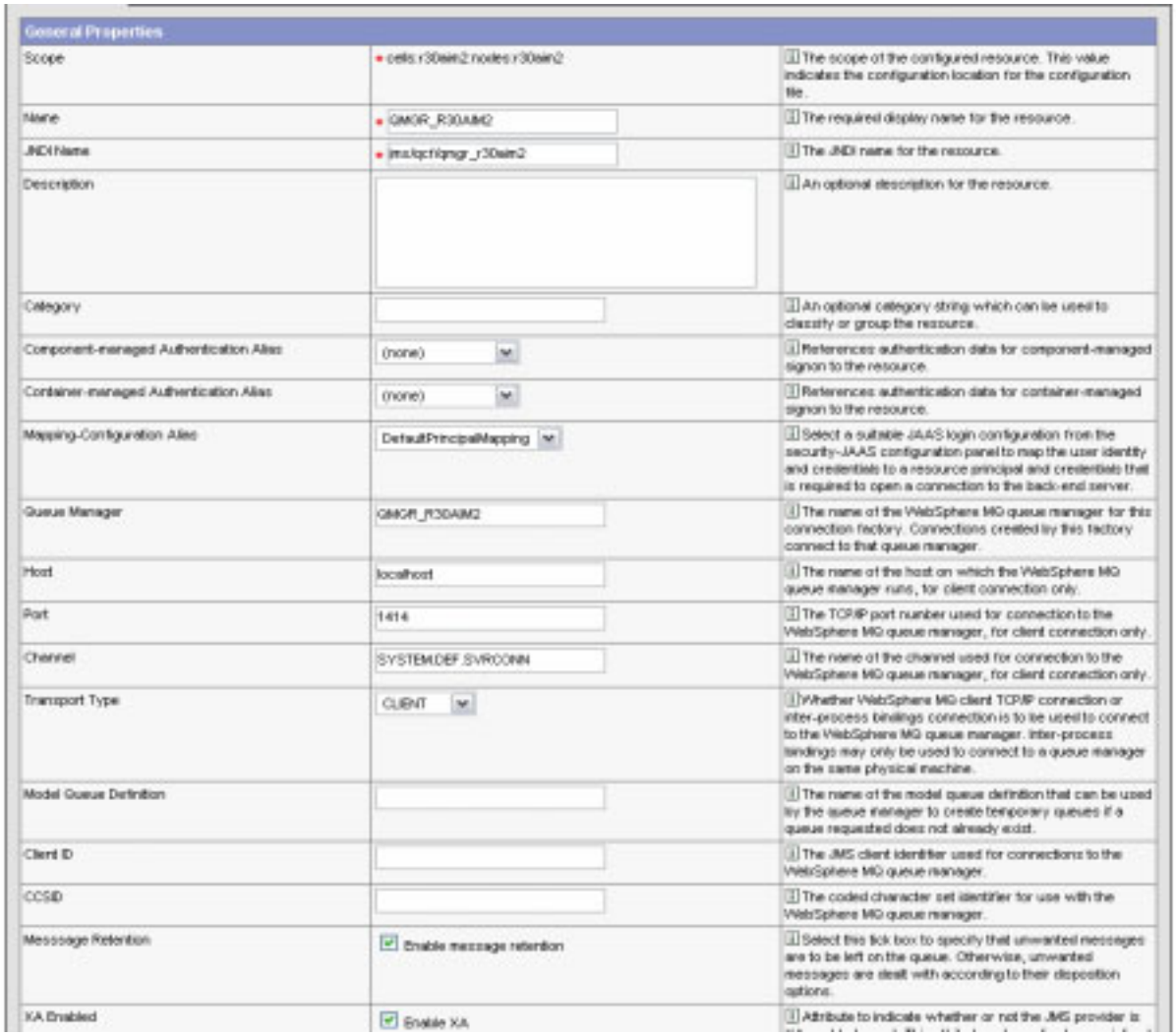


Figure 4-1: WebSphere New Queue Connection Factory

QueueDestination

Required for the Port-Listener. For each queue, request messages are received and a Port-Listener is defined, a QueueDestination has to be defined.

Sample for QueueDestination:

```
Resources > WebSphere MQ JMS Provider > WebSphere MQ Queue Destinations > New
```

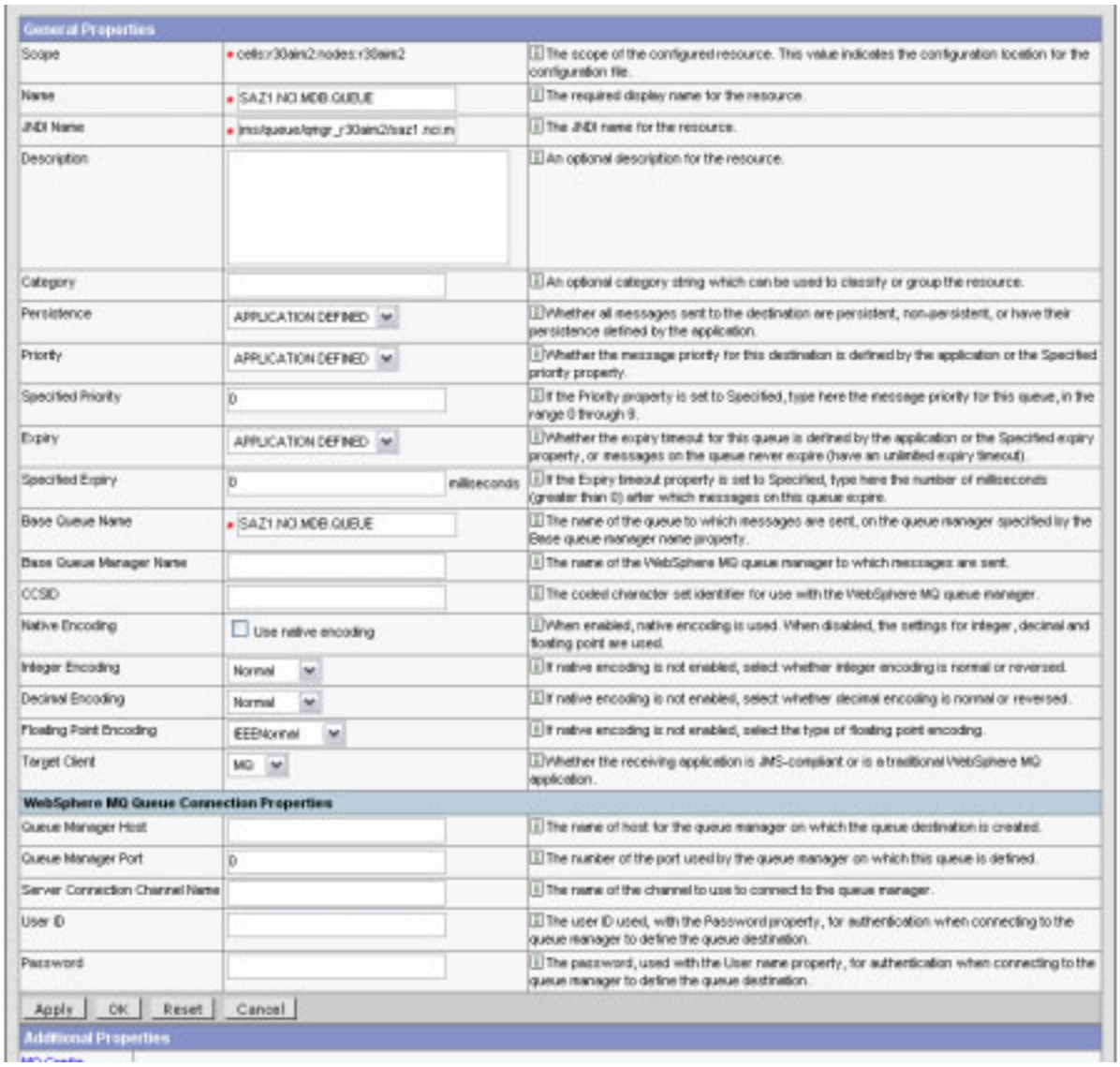


Figure 4-2: WebSphere New MQ Queue Destinations

Listener-Port

A Listener-Port has to be defined for each queue with incoming messages. This Listener-Port needs a predefined QueueConnectionFactory as well as a QueueDestination in the application servers JMS resource configuration.

Sample for Port-Listener:

```
Servers > Application Servers > server1 > Message Listener Service > New
```

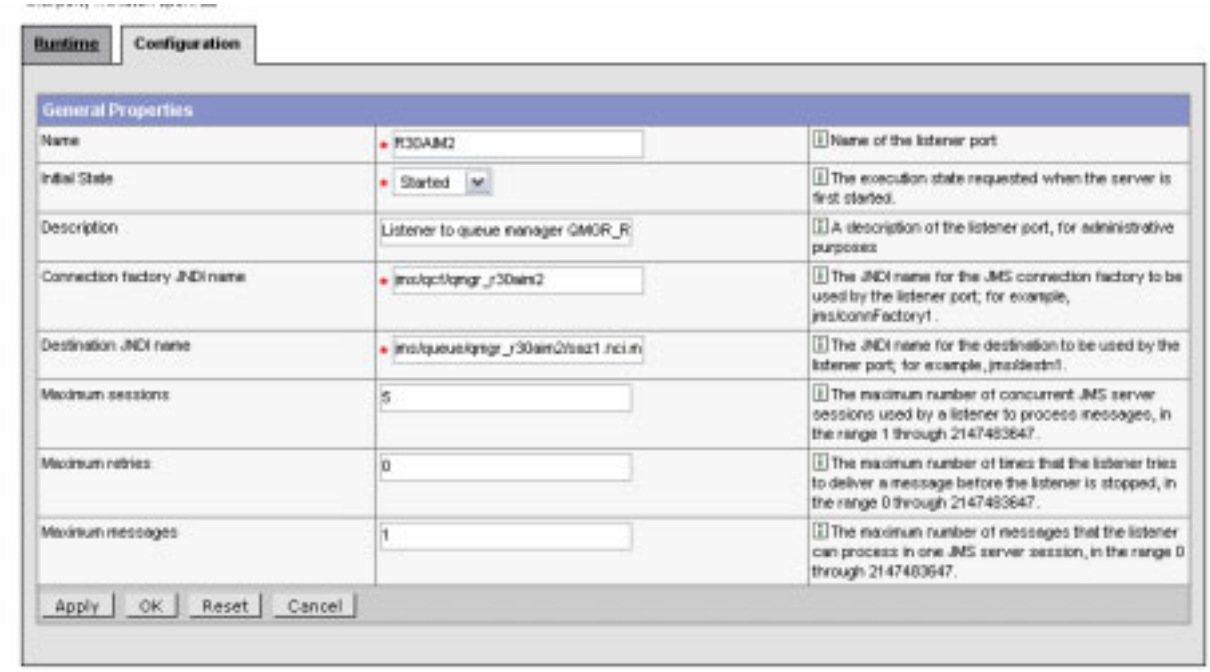


Figure 4-3: WebSphere New Message Listener Service

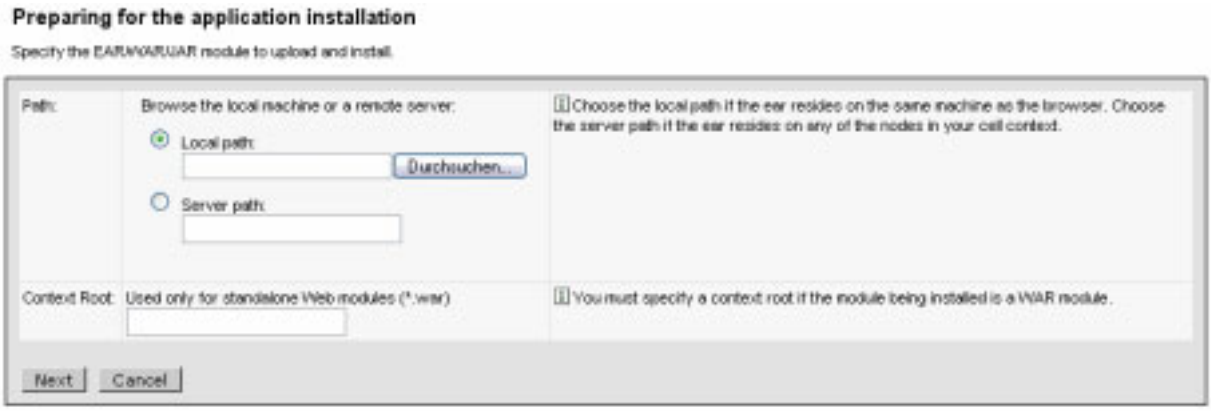
4.1.2 NciMDBAdapter deployment

Besides specifying the Port-Listener, the JNDI references defined in the NciMDBAdapter have to be mapped with the JNDI names of the required resources.

Sample for deploying the NciMDBAdapter:

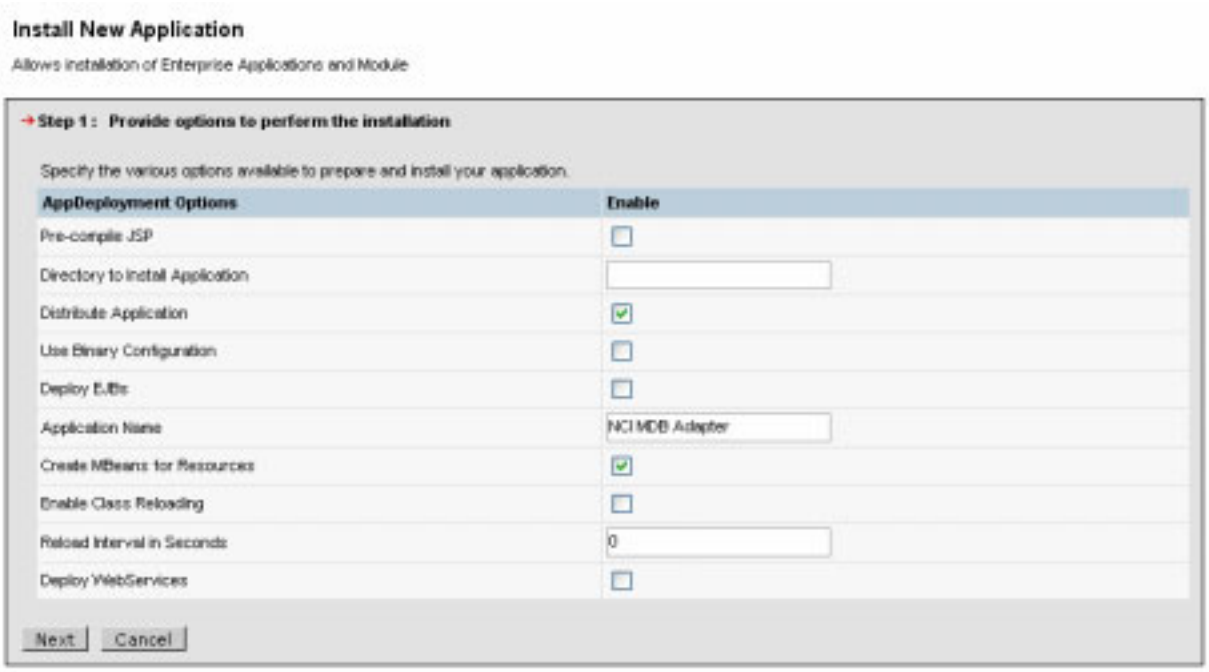
```
Applications > Install New Application
```

1. Select EAR-file to deploy, then continue

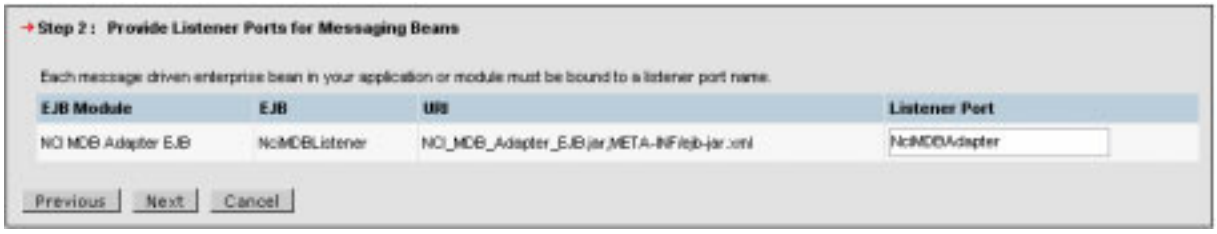


2. Continue with defaults

3. Select unique application name, then continue



4. Select Port-Listener, then continue

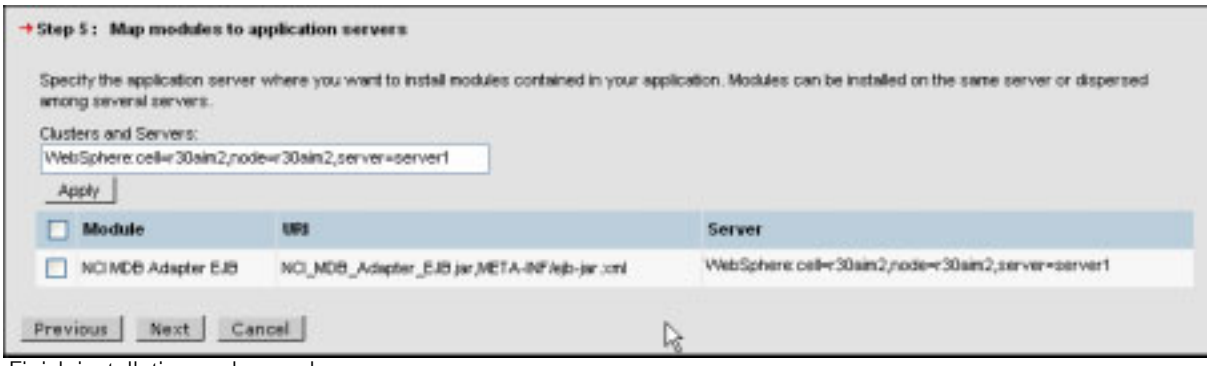


5. Enter JNDI name for message handler EJB, called by NciMDBAdapter to process messages, then

continue.

6. Select or enter JNDI name of QueueConnectionFactory used as connect queue manager for reply messages, then continue

7. Select Server for the EJB and continue (NCI MDB Pong EJB is no longer part of NciMDBAdapter EAR.



9. Finish installation and save changes.

4.1.3 Message handler programming guide

The NciMDBAdapter is passing the JMS message and if necessary an empty reply message to another bean which has to implement the NciMsgHandler and NciMsgHandlerHome interfaces. We call this the message handler. NciMDBAdapter comes with a sample message handler NciMDBPong, which replies to a text request message, by sending back the request message with “Reply from:” put in front of it.

HowTo write your own message handler:

To implement the interfaces NciMsgHandler and NciMsgHandlerHome, either the NciMDBHandler.EAR must be imported into your IDE workspace and your project has to refer to it, or the NciMsgHandler.JAR hast to be added to the classpath of your project.

The NciMsgHandler interface defines two methods:

```
public interface NciMsgHandler extends Remote {

        public Message processMessage(Message inMsg, Message replyMsg)
                throws java.rmi.RemoteException;

        public void processMessage(Message inMsg)
                throws java.rmi.RemoteException;
}
```

Figure 4-4: Nci MDB handler Remote interface

A. public Message processMessage(Message inMsg, Message replyMsg);
   This method is called if the received message is a request and requires a reply. An empty reply message, of the same type as the request message, is created by the NciMDBAdapter and passed to processMessage as the second parameter. The handler has to fill his content to reply into the reply message and give back the reply message as his return value.
B. public void processMessage(Message inMsg);
   This method is called if the received message is a datagram and does not requires a reply. The handler has just to process the message, no return value is given back.

Your application has to implement both methods, but if you do not expect either request messages or datagram messages, you can implement the corresponding method for example with an error message written to log and an exception thrown.

Remark: Exceptions thrown by either of the processMessage methods, are passed to the NciMDBAdapter and result in the Port-Listener being stopped if the maximum retry count is reached.

The NciMsgHandlerHome interface defines the create method:

```
public interface NciMsgHandlerHome extends EJBHome {
        /**
         * Creates a instance of a message handler bean implementing
         * the NciMsgHandler interface.
         */
        public NciMsgHandler create() throws CreateException, RemoteException;
}
```

Figure 4-5: Nci MDB handler Home interface

public NciMsgHandler create();
This method returns an instance of the message handler bean. The returned object is of type *NciMsgHandler*.
When creating an EJB e.g. with WebSphere Application Developer, the auto generated home interface already has a method create which returns an instance of the bean itself. This create method has to be removed; since the NciMDBAdapter isn't able to know the stub of all possible message handlers being developed. Instead the home interface has to return an instance of NciMsgHandler.

```
package com.tsystems.nci.mdb.handler.ejb;

import javax.ejb.SessionBean;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.TextMessage;

import com.ibm.ejs.ras.Tr;
import com.ibm.ejs.ras.TraceComponent;
import com.tsystems.nci.mdb.handler.NciMsgHandler;

/**
 * Bean implementation class for Enterprise Bean: NciMDBPongExt
 */
public class NciMDBPongExtBean implements SessionBean, NciMsgHandler {
	private static final TraceComponent tc =
		Tr.register(NciMDBPongExtBean.class);

	private javax.ejb.SessionContext mySessionCtx;
	/**
	 * getSessionContext
	 */
	public javax.ejb.SessionContext getSessionContext() {
		return mySessionCtx;
	}
	/**
	 * setSessionContext
	 */
	public void setSessionContext(javax.ejb.SessionContext ctx) {
		mySessionCtx = ctx;
	}
	/**
	 * ejbCreate
	 */
	public void ejbCreate() throws javax.ejb.CreateException {
	}
	/**
	 * ejbActivate
	 */
	public void ejbActivate() {
	}
	/**
	 * ejbPassivate
	 */
	public void ejbPassivate() {
	}
	/**
	 * ejbRemove
	 */
	public void ejbRemove() {
	}
	. . .
```

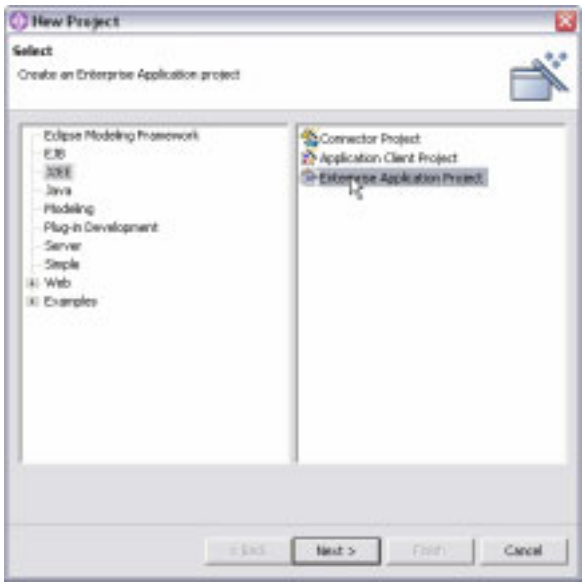
```
                } catch (JMSException e) {

                        . . .

                }
                Tr.exit(tc, "processMessage");
                return replyMsg;
        }

        /**
         * Process message, no result is returned to caller
         * @param msg Messageof type JMSMessage
         * @return String
         */
        public void processMessage(Message inMsg) {
                Tr.entry(tc, "processMessage");
                try {
                        StringBuffer response = new StringBuffer("Message received: ");
                        if (inMsg instanceof TextMessage) {
                                TextMessage inTextMsg = (TextMessage) inMsg;
                                if (inTextMsg.getText().startsWith("EXCEPTION")) {
                                        Tr.event(
                                                tc,
                                                "Exit handler with an exception has been
requested " +
                                                "by the sender, will force a
NULLPointerException");
                                        String ex = null;
                                        String res = ex.substring(1);
                                } else {
                                        response.append(inTextMsg.getText());
                                        System.out.println(response);
                                }
                        } else {
                                response.append("Message received not of type
'TextMessage'");
                                System.out.println(response);
                        }
                } catch (JMSException e) {
                        e.printStackTrace();
                        Tr.error(tc, "Error occured:" + e.getMessage());
                }
                Tr.exit(tc, "processMessage");
        }
}
```

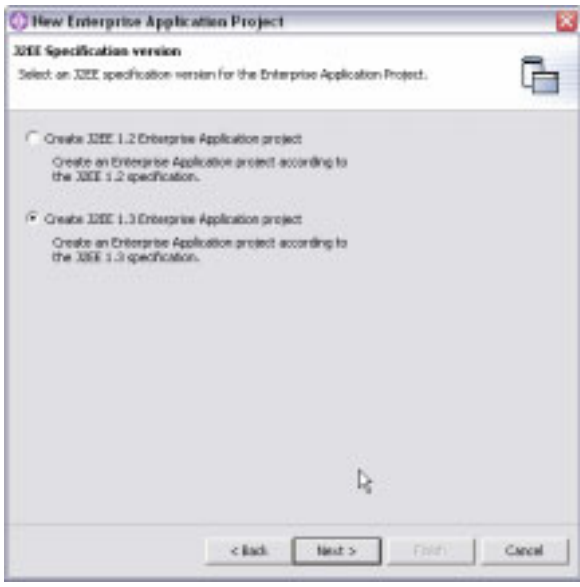
Figure 4-6: Sample Nci MDB handler implementation

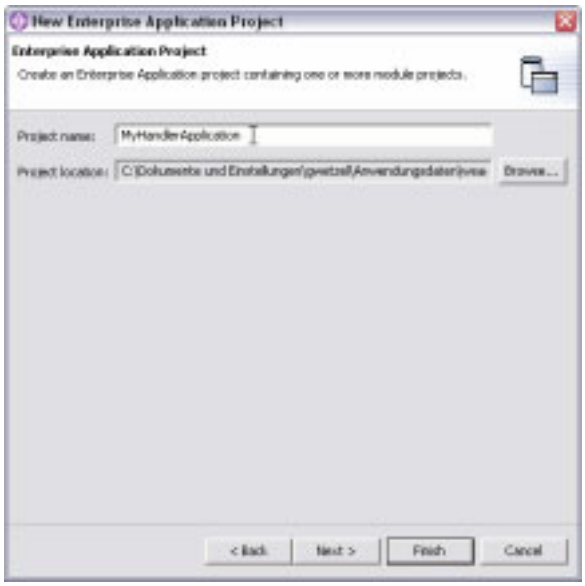
Step-by-Step creating a NciMDBHandler Bean in WebSphere Application Developer

To start developing a bean in WebSphere Application Develop, implementing the NciMsgHandler interfaces, follow these steps.

1. Create EAR Project including NciMsgHandler.JAR
   File > New > Project

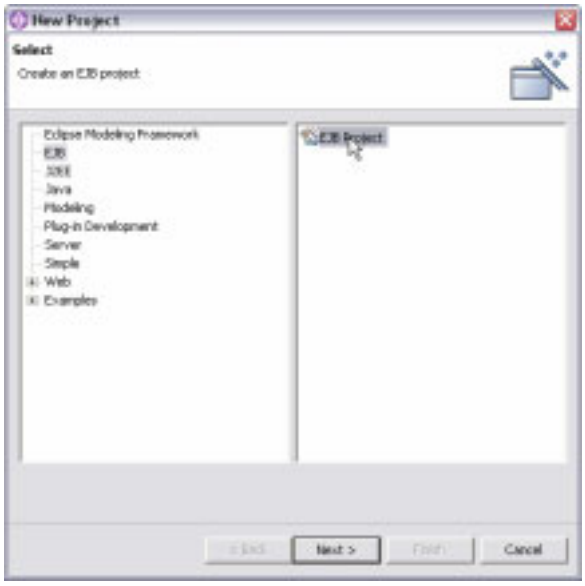


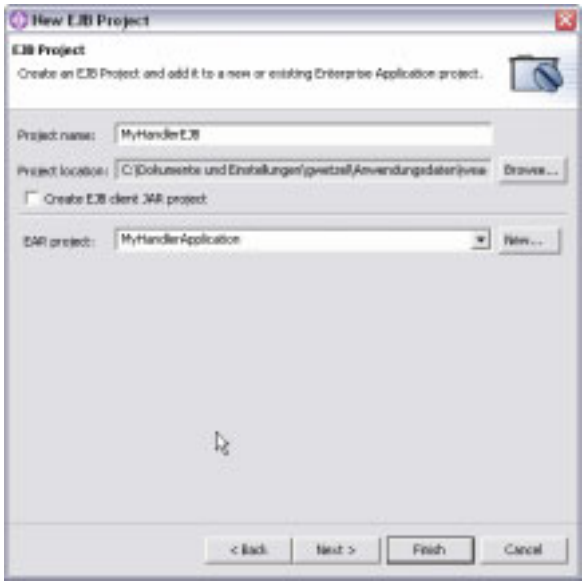


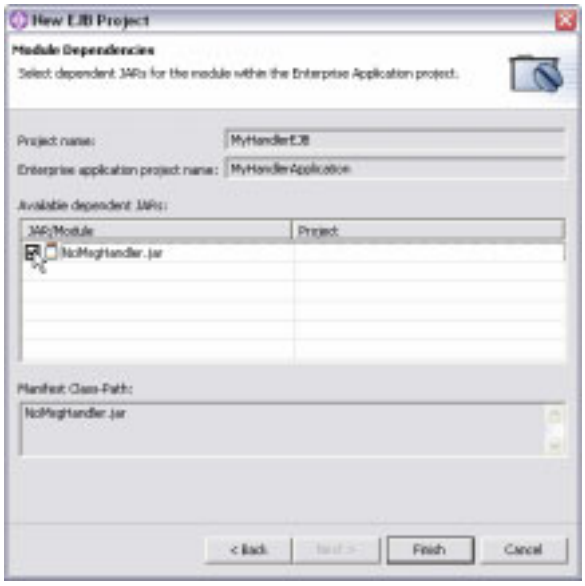


2. Import NciMsgHandler.jar into EAR project to ensure, NciMsgHandler.jar is in CLASSPATH
   Simply drag and drop the JAR into the EAR project.

3. Create EJB project and select NciMsgHandler.jar as dependent JAR
   File > New > Project

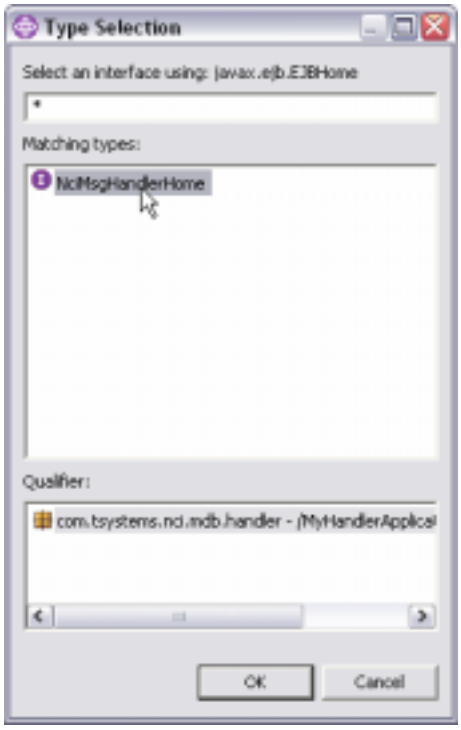


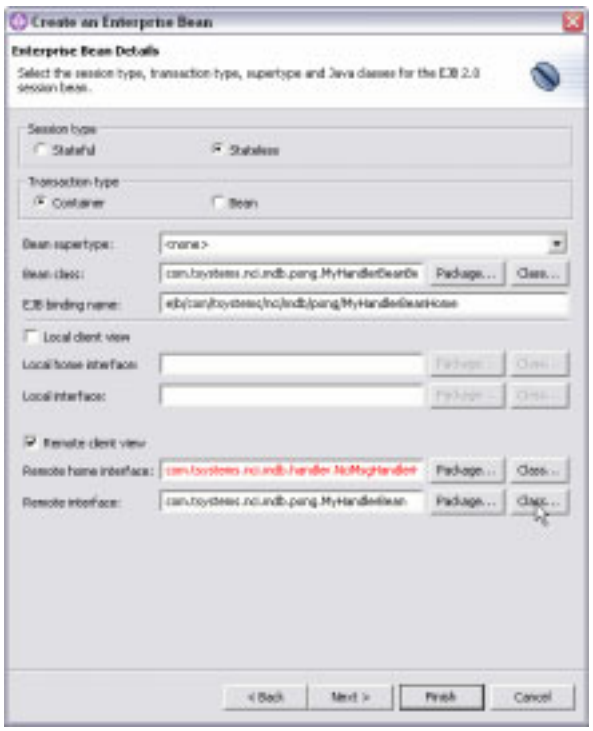


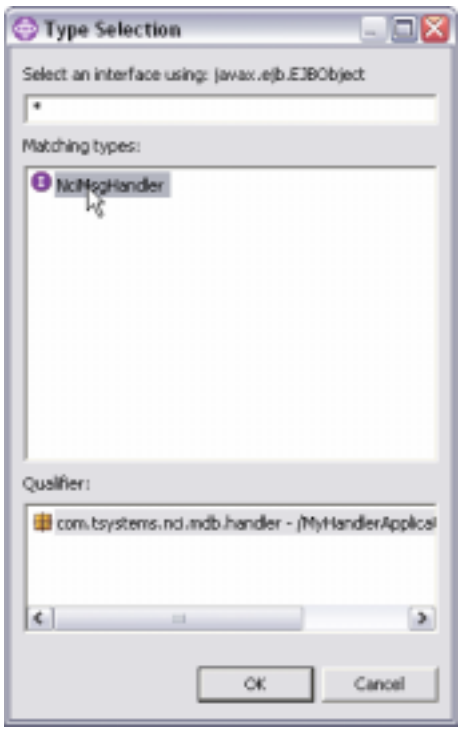


4. Create Enterprise Bean in EJB project
   File > New > Enterprise Bean

5. Select Remote Interface and Remote Home Interface



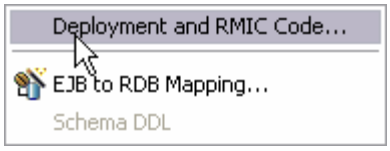


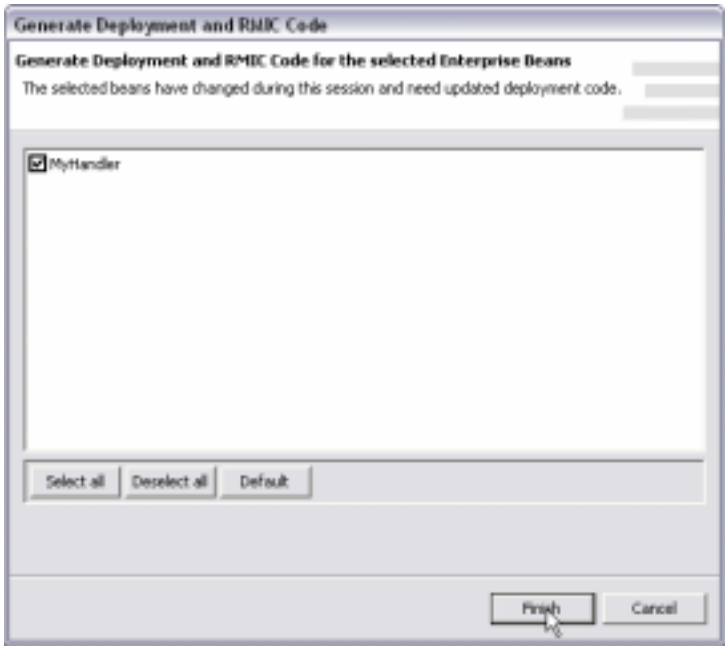


6. Implement processMessage( ) methods in new bean.

```
/**
 * Process JMS request message inMsg. Empty reply message of same type
 * as the request message is provided by caller. This message has to
 * be completed and returned to caller as return value.
 * @param inMsg JMS message
 * @param relyMsg empty Message of same type as inMsg
 * @return replyMsg
 */
public Message processMessage(Message inMsg, Message replyMsg) {
        /* YOUR CODING */
        . . .
}

/**
 * Process JMS datagram message, no result is returned to caller
 * @param msg Messageof type JMSMessage
 * @return void
 */
public void processMessage(Message inMsg) {
        /* YOUR CODING */
        . . .
}
```

7. Generate Deployment and RMIC code
   Popup Menu of EJB project > Generate > Deployment and RMIC Code





8. Create and deploy EAR file.

4.2 NciMDBPong – a Message Driven Beans Sample

A sample Enterprise Bean implementing the interfaces NciMsgHandler and NciMsgHandlerHome.This bean is only a simple sample to show how a bean could be developed implementing the application logic to process JMS messages passed on by the NciMDBAdapter.
The NciMDBPong can be deployed and it's JNDI name can be entered as reference for MyHandler in NciMDBAdapter. NciMDBAdapter will then call NciMDBPong to process the message given by the JMS Port-Listener.

The deployment is straight forward and no prerequisites are necessary.

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