Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms

Application Development Guide

Version 6.0

Note: Before using this information, and the product it supports, be sure to read the general information under *notices* on page 67

First edition (August 2005)
This edition applies to Rules and Formatter Extension for WebSphere [tm] Message Broker for Multiplatforms, Version 5.0, for IBM ® WebSphere Message Broker and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

At the back of this publication is a page titled “Sending your comments to IBM”. If you want to make comments, but the methods described are not available to you, please address them to:

IBM United Kingdom Laboratories
Information Development,
Mail Point 095,
Hursley Park,
Winchester,
Hampshire,
England,
SO21 2JN.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Chapter 1
Introduction

This chapter includes the following information:

- *About this Document*
- *Documentation Set*

About this Document

This guide provides descriptions and examples for each function of the Database Abstraction Layer.

This guide is organized into the following chapters:

Chapter 1, *Introduction*, provides a list of available documentation and documentation conventions.

Chapter 2, *Database Abstraction Layer APIs*, describes functions used in New Era of Networks Rules and Formatter APIs for database abstraction.

Documentation Set

The Rules and Formatter Extensions for WebSphere Message Broker for Multiplatforms documentation set includes:

- ***System Management Guide***
- ***New Era of Networks Formatter Programming Reference***
- ***New Era of Networks Rules Programming Reference***
- ***Application Development Guide***
- Rules, Formatter, and Visual Tester online help

- Installation Readme

Document Conventions

The following document conventions are used in this guide.

Text	Convention	Example
code	courier	<user ID> <password>
command line display	courier	`The message successfully parsed.`
command line entry	courier bold	**`NNFAD-t`**
command line prompt	courier	`Enter the input file name:`
path	regular	ora/bin (UNIX) ora\bin (NT)
book names	bold, italic	***Installation Guide***
chapter and section names	italic	*NT Installation*

Chapter 2

Application Programming

This chapter includes the following information:

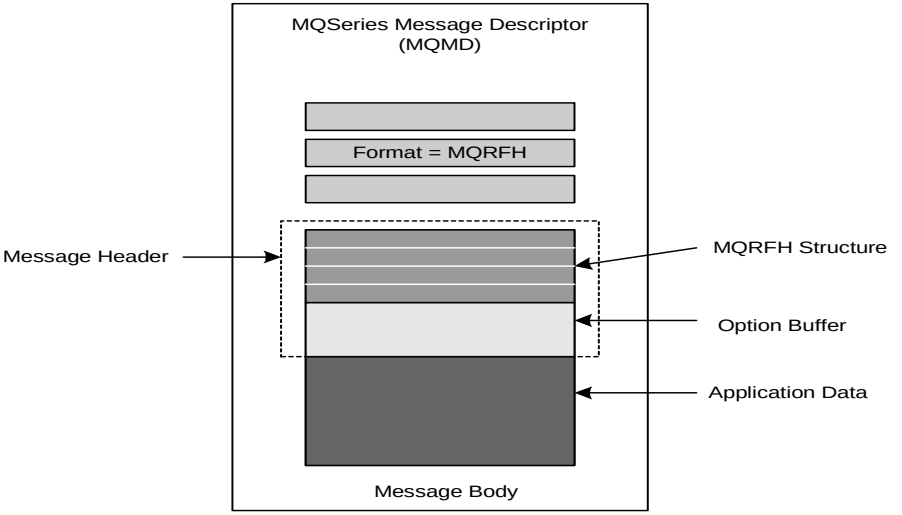
- *Message Processing*
- *Messages*

Message Processing

The WebSphere Rules broker combines MQSeries, New Era of Networks Formatter, and New Era of Networks Rules in a generic server process. The WebSphere Rules broker processes messages from one or more MQSeries input queues, uses New Era of Networks Formatter to parse messages, uses New Era of Networks Rules to determine what transformations to perform and where to route the messages, then puts the output messages on MQSeries queues for delivery to applications.

Messages

MQSeries messages sent to the rules processing broker have the following format:



WebSphere MQ Message Format

New Era of Networks Header

The first part of the message body is the New Era of Networks header. This header contains the application group and message type information that the New Era of Networks Rules processing broker requires to parse the message. If the New Era of Networks Rules processing broker receives a message that does not contain a New Era of Networks header, it assigns default values for both the application group and message type.

Applications that put messages to the New Era of Networks Rules processing broker input queue indicate that a New Era of Networks header precedes the

application data by setting the format field of the MQMD structure to MQFMT_RF_HEADER where MQFMT_RF_HEADER is defined as the eight-character string: MQHRFbbb (b = space).

The New Era of Networks header consists of two parts: the RFH2 structure and the New Era of Networks option buffer.

RFH2 Structure

The RFH2 structure contains the following fields:

Field	Description
StrucId (MQCHAR4)	The identifier for the RFH2 structure. The value must be: MQRFH_STRUC_ID = “RFHb” (b = space).
Version (MQLONG)	The identifier for Version-1 MQRFH structure. The value must be: MQRFH_VERSION_2.
StrucLength (MQLONG)	The length of the MQRFH structure and the subsequent option buffer. There is no default value for this field because the value depends on the length of the option buffer, which may be different for each message.
Encoding (MQLONG)	Encoding of the data following the MQRFH structure. The queue manager does not check the value of this field. The initial value of this field is MQENC_NATIVE.
CodedCharSetId (MQLONG)	Character set identifier of the data following the MQRFH structure. The queue manager does not check the value of this field. The initial value of this field is zero.
Format (MQCHAR8)	Format name of the data following the MQRFH structure. The queue manager does not check the value of this field. See the description of the Format field in the MQMD structure for more information about Format names. The initial value of this field is MQFMT_NONE.
Flags (MQLONG)	General flags.

New Era of Networks Option Buffer

The New Era of Networks option buffer immediately follows the RFH2 structure in the New Era of Networks header. It consists of a collection of space-delimited tag / value pairs:

<tagname1>b<value1>b<tagname2>b<value2>b (etc.)

The size of the New Era of Networks option buffer can be calculated using the following formula:

```
OptionBufferLength = MQRFH.StrucLength - sizeof(MQRFH)
```

Tag names and values cannot contain nulls or b (b = space).

Recognized names in option buffer:

- OPT_APP_GRP
  Application Group
- OPT_MSG_TYPE
  Message Type
- OPT_RELOAD_RULE_SET
  Reload Rule Set
- OPT_SHUTDOWN
  Shutdown

Note:
OPT_RELOAD_RULE_SET and OPT_SHUTDOWN cannot be used with WebSphere Message Brokers.

Chapter 3

Database Abstraction Layer APIs

This chapter includes the following information:

- *Overview*
- *APIs and Header Files*
- *Class Definitions*

Overview

This chapter describe the functions used in the New Era of Networks Rules and Formatter APIs for database abstraction. These Database Abstraction Layer APIs provide a means of managing transactions and maintaining data integrity.

You can ensure that the session state is accessible by using the Ok() function. OpenDbmsSession() provides the Rules and Formatter Extensions for WebSphere Message Broker for Multiplatforms functions with a session name to associate with the WebSphere Message Broker database. CloseDbmsSession() cleans up the session and releases any residual storage that may have been allocated by Rules and Formatter Extensions for WebSphere Message Broker for Multiplatforms or the DBMS during the execution of a program.

APIs and Header Files

Header files contain declarations for class functions and declarations for data types and constants.

Header Files

Object Class	Description	Header File
DbmsSession	Class DbmsSession declarations	ses.h
	Procedural APIs for DbmsSession	dbcapi.h

DbmsSession Object Methods

Return Type	Method	Arguments
int	IsUserHandle	N/A
void*	Handle	N/A
int	Ok	N/A
int	DbmsType	N/A
int	SetTCPNoDelay	int enable
int	connected	N/A
int	connect	N/A
int	reconnect	N/A
void	disconnect	N/A
int	BeginXact	(char* name = NULL)
int	RollbackXact	(char* name = NULL)
int	CommitXact	(char* name = NULL)
int	HasError	N/A

Return Type	Method	Arguments
int	GetErrCnt	N/A
const char*	GetErrNum	(int indx=0)
const char*	GetErrTxt	(int indx=0)
void	DumpErrors	(NNSY_NAMESPACE OStream)
void	ClearError	N/A
void*	SetError	(const char* const file, const int line)

DbmsSession Factory Functions

Return Type	Function	Arguments
DbmsSession*	OpenDbmsSession	(char *SessionName, int DbmsType)
DbmsSession*	OpenDbmsSession	(void* SessionHandle, int DbmsType)
DbmsSession*	OpenDbmsSession	(const char * const serverName, const char * const userID, const char * const passwd, const char * const dbInstance, int dbmsType)
DbmsSession*	OpenDbmsSession	(const char * const sessionName, const char* const configFileName, int DbmsType)
void	CloseDbmsSession	(DbmsSession* Session)

Transaction Layer Functions

Return Type	Function	Arguments
int	BeginXact	(DbmsSession* session, char* name = "")
int	RollbackXact	(DbmsSession* session, char* name = "")

Return Type	Function	Arguments
int	CommitXact	(DbmsSession* session, char* name = "")

Libraries

Shared libraries are archived collections of object files. Database Abstraction Layer libraries are located:

- In UNIX and z/OS, the libraries are in {installroot}/bin.
- In Windows, the shared libraries and DLLs are in {installroot}\bin.

For more information on shared libraries, refer to the example makefiles.

Class Definitions

The following functions are listed by class: DbmsSession member functions and nonmember DbmsSession factory functions, and then by functional categories: Accessors, Status, Transaction Management, and Error Handling.

DbmsSession Object

The DbmsSession is the first object you must allocate to work with New Era of Networks Rules, Formatter, and Messaging. All interaction with the database is performed through the DbmsSession object.

The DbmsSession object contains all the database connection information to execute queries against the database. The DbmsSession::Handle() method returns a pointer to the information you must pass to the individual database API routines for your database types.

DbmsSession contains a list of cumulative errors. The error handling functions allow you to view these errors. To prevent memory exhaustion failure, clear the error list periodically using ClearError(). See *ClearError* on page 43 for more information.

Syntax

```
class DbmsSession{
public:
DbmsSession()
DbmsSession(const char * const sessionName, intdbmsType);
DbmsSession(void *handle, intdbmsType);}
```

```
DbmsSession(const char * const sessionName,
            const char * const configFileName,
            intdbmsType):
DbmsSession(const char * const serverName,
            const char * const userId,
            const char * const passwd,
            const char * const dbInstance,
            intdbmsType):

virtual ~DbmsSession()

//Status information
virtual int Ok()
virtual int connect();
virtual int reconnect();
virtual void disconnect();

virtual int IsUserHandle()
virtual void* Handle();
virtual void *Handle() const
virtual NNOTSessionBase *getSession()
//The following function is not support by:
//Oracle
//MS SQL Server
//ODBC flavors

virtual int SetTCPNoDelay (int nd)

//Insert an error into the error list with id fileName and
//fileLine
virtual void SetError(const char* const fileName,
                      const int fileLine)

//Number of error codes encountered
virtual int GetErrCnt() const

//Specific error code in list
virtual const char* GetErrNum(int indx=0) const;

//Text describing errors
virtual const char* GetErrTxt(int indx=0) const;
```

```
//Clear all errors from session
virtual void ClearError ();


//Transaction Management
virtual int BeginXact(const char * const name = NULL);
virtual int RollbackXact(const char * const name = NULL);
virtual int CommitXact(const char * const name = NULL);
// DB information
virtual const char *getUserID() const;
virtual const char *getServer() const;
virtual const char *getDBinst() const;
virtual const char *getDBversion() const;
```

DbmsSession Factory Functions

OpenDbmsSession

OpenDbmsSession searches the nnsyreg.dat configuration file for an entry named SessionName and instantiates a DbmsSession object of type DbmsType. The nnsyreg.dat file must be in the same directory as the program executable file.

Syntax

```
DbmsSession* OpenDbmsSession(char *SessionName, int DbmsType);
```

Parameters

Name	Type	Input/ Output	Description
SessionName	char *	Input	Identifies the session tag name to be used in the configuration file. The tag name is the first field of the configuration sequence in the configuration file.
DbmsType	int	Input	Identifies the type of database. Supported types are defined in dbtypes.h and in the *Database Session Types* on page 25

Remarks

A call to OpenDbmsSession() is required prior to using any New Era of Networks Rules, Formatter, and Messaging APIs. Make sure the session state is accessible using Ok(). Ok() returns 1 if the session state is operational.

Return Value

Returns session pointer to an object of type DbmsSession for use in other WebSphere Message Broker API calls.

Example

```
...
DbmsSession *mySession;
mySession = OpenDbmsSession("mytag",NN_DB_TYPE_ORA8);
if ( !mySession || !mySession->Ok() ) {
              ... /* Database session not created or not
                      connected */
}
... /* Use for Queuing, Rules, or Formatter */
```

See Also

OpenDbmsSession (SessionHandle, DbmsType)

OpenDbmsSession (serverName, userID, passwd, dbInstance, DbmsType)

OpenDbmsSession (SessionName, configFileName, DbmsType)

CloseDbmsSession ()

OpenDbmsSession

OpenDbmsSession() allows users to construct a DbmsSession from an existing, database-specific, user-created session handle. DbmsType indicates the database product and version. For more information, see *Database Session Types* on page 25.

Syntax

```
DbmsSession* OpenDbmsSession(void* SessionHandle,
                             int DbmsType);
```

Parameters

Name	Type	Input/ Output	Description
SessionHandle	void*	Input	Database specific connection handle.
DbmsType	int	Input	Supported types are defined in dbtypes.h and in the *Database Session Types* on page 25

Remarks

A call to OpenDbmsSession() is required prior to using any of the New Era of Networks Rules, Formatter, and Messaging APIs. Make sure the session state is accessible using Ok(). Ok() should return 1 if the session state is operational.

Return Value

Returns a session pointer to an object of type DbmsSession for use in other WebSphere Message Broker API calls.

Example

```
...
#include "dbtypes.h"
DbmsSession *mySession;
OCISvcCtx *mySvcCtx;
... // Manually logon to Oracle database
mySession = OpenDbmsSession((void *)mySvcCtx,NN_DB_TYPE_ORA8);
if ( !mySession || !mySession->Ok() ) {
               ... // Database session not created or not
connected
}
... // Use for Messaging and Queuing, Rules or Formatter
```

See Also

OpenDbmsSession (SessionName, DbmsType)

OpenDbmsSession (serverName, userID, passwd, dbInstance, DbmsType)

OpenDbmsSession (SessionName, configFileName, DbmsType)

CloseDbmsSession ()

OpenDbmsSession

This overloaded version of OpenDbmsSession() enables you to connect to a database using an existing database server name, user ID, password, database instance, and database type. DbmsType indicates the database product and version. For more information, see *Database Session Types* on page 25.

Syntax

```
DbmsSession* OpenDbmsSession(const char * const serverName,
                             const char * const userID,
                             const char * const passwd,
                             const char * const dbInstance,
                             int DbmsType);
```

Parameters

Name	Type	Input/ Output	Description
serverName	const char * const	Input	Server on which the database resides.
userId	const char * const	Input	Database user name.
passwd	const char * const	Input	Database password.
dbInstance	const char * const	Input	Specific instance of database. For Sybase and SQL Server only.
DbmsType	int	Input	Type of database used. See the *Database Session Types* on page 25

Remarks

A call to OpenDbmsSession() is required prior to using any WebSphere Message Broker APIs. Make sure the session state is accessible using Ok(). Ok() should return 1 if the session state is operational.

Return Value

Returns session pointer to an object of type DbmsSession for use in other API calls.

Example

```
...
#include "dbtypes.h"
DbmsSession *mySession;
mySession = OpenDbmsSession("A","B","C", "D",NN_DB_TYPE_ORA8);
if ( !mySession || !mySession->Ok() ) {
              .../* Database session not created or not
connected */
}
... /* Use for Messaging and Queuing, Rules, or Formatter */
```

See Also

OpenDbmsSession (SessionName, DbmsType)

OpenDbmsSession (SessionHandle, DbmsType)

OpenDbmsSession (SessionName, configFileName, DbmsType)

CloseDbmsSession ()

OpenDbmsSession

Use this call to open a DbmsSession with a specific file other than nnsyreg.dat.

Syntax

```
OpenDbmsSession (const char* const sessionName,
                 const char* const configFileName,
                 int DbmsType)
```

Parameters

Name	Type	Input/ Output	Description
SessionName	const char* const	Input	Identifies the session tag name in the configuration file. The tag name is the first field of a line in the configuration file.
ConfigFileName	const char* const	Input	The configuration file name that has the same format as the default nnsyreg.dat file name.
DbmsType	int	Input	Identifies the type of database. Supported data types are defined in dbtypes.h and in the *Database Session Types* on page 25

Remarks

The alternative configuration file must be in the same format as the nnsyreg.dat file.

Return Value

Returns a session pointer for use in other API calls; NULL if unable to associate the provided information with a session.

Example

```
...
DbmsSession *session = OpenDbmsSession
   ("oraHub",
    "configFile.txt", NN_DBMS_TYPE_ORA8);
    if (!session)
      // handle error
```

See Also

OpenDbmsSession (SessionName, DbmsType)

OpenDbmsSession (SessionHandle, DbmsType)

OpenDbmsSession (serverName, userID, passwd, dbInstance, DbmsType)

CloseDbmsSession ()

CloseDbmsSession

CloseDbmsSession() cleans up a New Era of Networks Rules, Formatter, and Messaging session and releases any residual storage that might have been allocated by New Era of Networks Rules, Formatter, and Messaging during execution. Once a session is closed, use OpenDbmsSession() to establish another DBMS session.

Syntax

```
void CloseDbmsSession(DbmsSession* Session);
```

Parameters

Name	Type	Input/ Output	Description
Session	DbmsSession*	Input	Pointer to an open DbmsSession. Session must have been allocated using one of the OpenDbmsSession() methods.

Remarks

Call CloseDbmsSession() after completing all New Era of Networks Rules, Formatter, and Messaging processing.

Return Value

None. There are no error-handling functions for CloseDbmsSession().

Example

```
... // All work on open session mySession is complete
CloseDbmsSession(mySession);
mySession = NULL;
```

See Also

OpenDbmsSession (SessionName, DbmsType)

OpenDbmsSession (SessionHandle, DbmsType)

OpenDbmsSession (serverName, userID, passwd, dbInstance, DbmsType)

OpenDbmsSession (SessionName, configFileName, DbmsType)

DbmsSession Member Functions

Ok

The state of the DbmsSession class.

Syntax

```
virtual int Ok();
```

Parameters

None.

Remarks

None.

Return Value

Returns 1 or TRUE for a class state that is usable and zero (0) or FALSE if the class is not usable.

Example

```
...
DbmsSession *mySession;
... // logon to the Sybase database
mySession = OpenDbmsSession("nnsySYB",NN_DB_TYPE_SYB_CT);
if ( !mySession || !mySession->Ok() ) {
... // Database session not created or not connected
}
```

DbmsType

Identifies the database type of DbmsSession object.

Syntax

```
virtual int DbmsType();
```

Parameters

None

Return Value

Returns the DBMS type of DbmsSession. The NN_DB_TYPE macro is set according the database type and provides the appropriate value. The DBMS types are defined in dbtypes.h. The dbtypes.h file must be included in the header file.

Database Session Types

Return Value	Description
NN_DB_TYPE_SYB_CT	Sybase ctlib
NN_DB_TYPE_ORA8	Oracle 8.0.X, 9.X, 10.X
NN_DB_TYPE_MSSQL	Microsoft SQLServer
NN_DB_TYPE_DB2	IBM DB2 ODBC CLI
NN_DB_TYPE_ODBC	ODBC
NN_DB_TYPE_MQSERIES	IBM MQSeries

Note:
Effective with New Era of Networks Rules, Formatter, and Messaging release 5.0, database session names have changed. Earlier DbmsType names are supported, but are no longer accurate for use with New Era of Networks Rules, Formatter, and Messaging Release 5.0 and later.

Example

```
...
if (mySession->DbmsType() == NN_DB_TYPE_SYB_CT) {
  myHandle = (CS_CONNECTION)mySession->Handle();
}      ...
```

See Also

Handle ()

OpenDbmsSession (sessionHandle, DbmsType)

IsUserHandle

Returns TRUE if the SessionHandle for this session is provided by the user.

Syntax

```
virtual int IsUserHandle()
```

Parameters

None

Example

```
...
DbmsSession *mySession=newDbmsSession((void*)lda,
                                      NN_DB_TYPE_ORA8);
...
if(mysession->IsUserHandle())
```

Handle

Returns a void pointer to a database handle. Use this handle to execute database API functions directly, using the connection from the current DbmsSession object.

Syntax

```
virtual void* Handle() const
virtual void* Handle()
```

Parameters

Handle Type	Database	Structure
DBPROCESS*	MSSQL Server	Pass this handle to Sybase dblib routines.
CS_ CONNECTION*	Sybase ctlib	Pass this handle to Sybase ctlib routines. CS_CONNECTION*ctlibptr= (CS_CONNECTION*) session→Handle();

struct OraSesHandle*	Oracle	struct Ora8SesHandle {
		OCIServer *serverH;
		OCIError *errorH;
		OCIEnv *envH;
		OCISvcCtx *SvcCtxH;
		OCISession *sessionH;
		};
		Use members with OCI routines.

ODBCHandle_t*	DB2	typedef struct ODBCSesHandle { HENV hEnv; HDBC hDbc; HSTMT hStmt; } ODBCHandle_t; Use the hEnv, hDbc, hStmt members with ODBC routines.

Handle Type	Database	Structure
ODBCHandle_t*	ODBC	typedef struct ODBCSesHandle { HENV hEnv; HDBC hDbc; HSTMT hStmt; } ODBCHandle_t; Use the hEnv, hDbc, hStmt members with ODBC routines.

Example

```
...
if(mySession->DbmsType() ==NN_DB_TYPE_MSSQL) {
                  DBPROCESS *MyDBProc=(DBPROCESS*)mySession-
>Handle();
}              ...
```

connect

Call this function to connect the session to the database. By default, a session is connected when it is created. You should only need to use this call if you explicitly call disconnect().

Syntax

```
virtual int connect();
```

reconnect

If the session is connected to a database, calling this function will disconnect the session from the database. If successful, the function will then try to connect back to the database. This call is equivalent to calling disconnect() followed by connect().

If the session is not connected to a database, calling this function will connect to the database using the information passed when the session was created.

Note:
Calling this function could cause undetermined behavior if the session is being used by another New Era of Networks instance.

Syntax

```
virtual int reconnect();
```

disconnect

This function disconnects the session from the database.

Syntax

```
virtual int disconnect();
```

Note:
Calling this function could cause undetermined behavior if the session is being used by another New Era of Networks instance.

getSession

Reserved for internal use.

Syntax

```
virtual NNOTSessionBase *getSession()
```

SetTCPNoDelay

SetTCPNoDelay() enables the user to modify the TCP message aggregation algorithm for the socket-supporting communication between the database client and server. This function is only necessary in a few situations. For example, if your application resides on a system relative to the system hosting the database server, and if you are having messaging performance problems.

Syntax

```
int SetTCPNoDelay(int onOff);
```

Parameters

Name	Type	Input/ Output	Description
onOff	int	Input	Boolean value; zero (0) = off, 1 = on.

Remarks

This is a Sybase-specific function. Calling this function with the value of 1 causes every message sent to the server to be sent immediately, regardless of the size of the message. Poor network performance can result.

Return Value

Returns zero (0) on success; -1 if not supported or on failure.

There are no error-handling functions for SetTCPNoDelay().

Example

```
...
DbmsSession *mySession;
... // logon to the Sybase database
mySession = OpenDbmsSession("session name",NN_DB_TYPE_SYB_CT);
if ( !mySession || !mySession->Ok() ) {
          ... // Database session not created or not connected
}
// Set TCP_NODELAY
if (mySession->SetTCPNoDelay (1) < 0) {
  StandardErrorStream << "Unable to set TCP no delay" << endl;
}
```

HasError

Query for any DBMS error that occurred during processing.

Syntax

```
virtual int HasError ();
```

Parameters

None.

Remarks

None.

Return Value

Returns 1 or TRUE if errors have occurred; zero (0) or FALSE if no errors have occurred.

Example

```
...
DbmsSession *mySession;
mySession = OpenDbmsSession("sybHP",NN_DB_TYPE_SYB_CT);
if ( mySession && mySession->Ok() ) {
    ...   // <code that could generate errors>
    if (mySession->HasError())
        mySession->DumpErrors(StandardErrorStream);
}
```

See Also

GetErrCnt ()

GetErrNum ()

GetErrTxt ()

DumpErrors ()

ClearError ()

SetError ()

SetError

SetError transfers any error information contained in the database handle structure into the error number and error text storage of the DbmsSession object. This function should only be used by application programmers who execute direct database API calls (Sybase CTLIB, Sybase / MSSQL dblib, Oracle OCI, or DB2 ODBC). SetError is used to log database errors into the session object after a database API call returns a failure code.

Syntax

```
virtual void SetError(const char* const fileName,
                      const int fileLine)
```

Parameters

Name	Type	Input / Output	Description
fileName	const char*	Input	Name of file where error occurred.
fileLine	const char*	Input	Line where error occurred.

Remarks

This function allows you to specify the filename and file line from which the error was logged. Pass the file argument using the __FILE__ preprocessor macro, and pass the line argument using the __LINE__ macro.

If you are making database API calls directly in your code, check the return code of ALL database API calls. If an error is detected, call SetError(). You do not have to pass in error information. This information is retrieved from the database connection handle in the session object.

Return Value

None.

See Also

HasError ()

GetErrCnt ()

GetErrNum ()

GetErrTxt ()
DumpErrors ()
ClearError ()

GetErrCnt

During processing, more than one DBMS error can occur. GetErrCnt returns the number of errors stored for extraction by the user. With subsequent calls to the database, a new error count begins.

Syntax

```
virtual int GetErrCnt ()const
```

Parameters

None.

Remarks

None.

Return Value

The number of errors contained in the error list.

Example

```
...
DbmsSession *session;
session = OpenDbmsSession("sybHP",NN_DB_TYPE_SYB_CT);
    ...
if (session->HasError())
  for (int inx=0; inx < session->GetErrCnt(); inx++)
    printf("DbmsSession ERROR_NUM=%d, ERROR_TEXT=%s",
        session->GetErrNum(inx),session->GetErrTxt(inx));
```

See Also

HasError ()

GetErrNum ()

GetErrTxt ()

DumpErrors ()

ClearError ()

SetError ()

GetErrNum

When DBMS errors occur, the database provides error identification numbers. You can obtain these error numbers through this member function.

Syntax

```
virtual const char *GetErrNum (int indx=0)const
```

Parameters

None. When passed with no parameters, the last error number is returned.

Remarks

None.

Return Value

The DBMS error number for the error item contained in the error list.

Example

The following example uses inx to walk through a list of the set of recent errors returned by the GetErrCnt() function. GetErrTxt(inx) returns the matching text in the for loop. inx is an abstract integer that refers to an index into a list of errors available to the application programmer.

```
...
DbmsSession *session;
session = OpenDbmsSession("sybHP",NN_DB_TYPE_SYB_CT);
    ...
if (session->HasError())
  for (int inx=0; inx < session->GetErrCnt(); inx++)
    printf("DbmsSession ERROR_NUM=%d, ERROR_TEXT=%s",
        session->GetErrNum(inx),session->GetErrTxt(inx));
```

See Also

HasError ()

GetErrCnt ()

GetErrTxt

When DBMS errors occur, the database provides an error description. Use this member function to obtain the error description.

Syntax

```
virtual const char *GetErrTxt (int indx=0)const
```

Parameters

Single integer parameter that specifies the error to return information about. When passed with no parameters, text for the last error is returned.

Remarks

None.

Return Value

Returns the DBMS error text for the error items contained in the error list.

Example

The following example uses int to walk through a list of the set of recent errors returned by the GetErrCnt() function. GetErrTxt(inx) returns the matching text in the for loop. inx is an abstract integer that refers to an index into a list of errors available to the application programmer.

```
...
DbmsSession *session;
session = OpenDbmsSession("sybHP",NN_DB_TYPE_SYB_CT);
    ...
if (session->HasError())
  for (int inx=0; inx < session->GetErrCnt(); inx++)
    printf("DbmsSession ERROR_NUM=%d, ERROR_TEXT=%s",
        session->GetErrNum(inx),session->GetErrTxt(inx));
```

See Also

HasError ()
GetErrCnt ()
GetErrNum ()
DumpErrors ()
ClearError ()
SetError ()

ClearError

ClearError removes any error information stored in the session object from previous database operations. This function is the only way to clear database errors from the session. After an error occurs, the application programmer must call this function to continue using the session.

Syntax

```
virtual void ClearError()
```

Parameters

None.

Return Value

None.

Example

```
...
mySession->ClearError()
```

See Also

HasError ()

GetErrCnt ()

GetErrNum ()

GetErrTxt ()

DumpErrors ()

SetError ()

DumpErrors

This function is only called if HasError() returns TRUE. DumpErrors() loops through all errors stored on the session, in order, and prints the error number and description for each error to the provided output stream (os argument).

DumpErrors() simplifies the task of reporting multiple database errors. You can also use GetErrCnt(), GetErrNum(), and GetErrTxt().

Syntax

```
virtual void DumpErrors(NNSY_NAMESPACE OStream &os) const
```

Parameters

Name	Type	Input/ Output	Description
os	OStream &	Input	output stream, for example, StandardOutStream or StandardErrorStream

Return Value

None.

Example

The following error occurred on a Sybase (dblib) connection and was caused by an improper user id. Sybase reported two distinct errors, and the output was created with a single call to DumpErrors().

```
Error: 4002
Text:  ------ file: /home/.../trunk/sqlsvr/sqlses.cpp  line:
967 ------
DBLIB MSG: [msgno] 4002 [severity] 14 [msg] Login failed.
[server] RODAN
Error: 20014
Text:  ------ file: /home/.../trunk/sqlsvr/sqlses.cpp  line:
967 ------
DBLIB ERR: [dberrno] 20014 [db error] Login incorrect.
[severity] 2
```

See Also

HasError ()
GetErrCnt ()
GetErrNum ()
GetErrTxt ()
ClearError ()
SetError ()

BeginXact

BeginXact() begins a database transaction. The DBMS assigns a unique internal number used to track all work done within a bracketed transaction.

Syntax

```
virtual int BeginXact (const char * const name = NULL)
```

Parameters

Name	Type	Input/ Output	Description
name	const char * const	Input	Optional begin modifier. Some databases allow named transactions for nesting purposes. Defaults to "".

Remarks

Some databases allow the begin transaction statement to have a modifier. Although it is supported, using the optional name parameter is not recommended.

Return Value

Returns a nonzero value after successfully beginning a transaction; zero (0) on failure.

There are no error-handling functions for BeginXact().

Example

```
...
DbmsSession *mySession;
... // Open database connection, other setup
If(!mySession->BeginXact()){
//We have an error condition that needs handling
}
```

See Also

CommitXact ()

RollbackXact ()

RollbackXact

RollbackXact() performs a rollback on all work done prior to the last commit. This function allows for abstraction between DBMS products and features.

Syntax

```
virtual int RollbackXact(const char * const name = NULL);
```

Parameters

Name	Type	Input/ Output	Description
name	const char * const	Input	Optional rollback modifier. Some databases allow named transactions for nesting purposes. Defaults to "".

Remarks

DBMS applications use logical units of work and allow recovery schemes to either apply or remove the information using the commit or rollback features for the DBMS.

Some databases allow the rollback statement to have a modifier and enable multiple transaction boundaries. Although it is supported, using the optional rollback modifier is not recommended.

Return Value

Returns a nonzero integer value if rollback is successful; zero (0) on failure.

There are no error-handling functions for RollbackXact().

Example

```
...
DbmsSession *mySession;
... // Open database connection, other setup
VQueue * myQueue = CreateQueue("MyQ",mySession,0);
char *myMessage = "This is a test message";
if (!mySession->BeginXact() ) {
// Some operation, such as queue->put()
if ( !queue->put((void *)myMessage,strlen(myMessage)) ) {
    // Roll back the work done
    mySession->RollbackXact();
}
```

See Also

BeginXact ()

CommitXact ()

CommitXact

CommitXact() performs a commit on all work done within the current transaction bracket. This function allows you to make the specific DBMS implementation of this feature abstract, so that it is more portable.

Syntax

```
virtual int CommitXact(const char * const name = NULL)
```

Parameters

Name	Type	Input/ Output	Description
name	const char * const	Input	Optional commit modifier. Some databases allow named transactions for nesting purposes. Defaults to "".

Remarks

DBMS implementations use the commit command to move the work done within the transaction bracket in the physical database. Prior to doing a commit, the work is not available to other processes that would usually access the database tables involved in the transaction.

Some databases allow the commit statement to have a modifier and enable multiple transaction boundaries. Although it is supported, using the optional rollback modifier is not recommended.

Return Value

Returns a nonzero integer value if a message is committed successfully; zero (0) on failure.

If CommitXact() fails, then use HasError().

Example

```
...
DbmsSession *mySession;
... // Open database connection, other setup
VQueue * myQueue = CreateQueue("MyQ",mySession,0);
char *myMessage = "This is a test message";
if ( !mySession->BeginXact() ) {
if ( !queue->put((void *)myMessage,strlen(myMessage)) ) { //
Some operation, such as queue->put()
      mySession->RollbackXact();// Roll back the work done
}
else
{
      mySession->CommitXact(); // operation successful, commit
work
}
```

See Also

BeginXact ()

RollbackXact ()

getUserID

Returns the user name used when connecting to the database.

Syntax

```
virtual const chat *getUserID() const;
```

getServer

Returns the name of the server this session connects to.

Syntax

```
virtual const chat *getServer() const;
```

getDBinst

Returns the name of the database this session connects to

Syntax

```
virtual const chat *getDBinst() const;
```

Note:
This is only valid for Sybase and MS SQL server sessions.

getDBversion

Returns the version string of the database server associated with this connection.

Syntax

```
virtual const chat *getDBversion() const;
```

Chapter 4
buildMessage

The buildMessage routine builds messages with an WebSphere Message Broker header and initializes the associated message descriptor. After calling buildMessage, the application can call MQPUT with the message descriptor and the message buffer supplied by buildMessage.

Note:
SHUTDOWN and RELOAD are not applicable to WebSphere Message Brokers.

Function Declaration for buildMessage

```
int buildMessage(MQMD* md,
                 char* data,
                 long dataLength,
                 char *dataFormat,
                 int *bufferLength,
                 char *buffer,
                 char *applicationGroup,
                 char *messageType,
                 int shutdown,
                 int reload)
```

Parameter Descriptions for buildMessage

Parameter	Description
md	Pointer to an MQSeries message descriptor allocated by the calling application.
data	Pointer to the application data.
dataLength	Length of the application data.

Parameter	Description
dataFormat	The format of the data contained in the buffer pointed to by the data parameter.
buffer	The pointer to the memory where buildMessage puts the WebSphere Message Broker message.
bufferLength	The size of the buffer.
applicationGroup	The application group to associate with the message. This parameter should be set to NULL when building SHUTDOWN messages. **Note**: SHUTDOWN is not applicable to WebSphere Message Brokers.
messageType	The message type to associate with the message. This parameter should be set to NULL when building SHUTDOWN messages. **Note**: SHUTDOWN is not applicable to WebSphere Message Brokers.
shutdown	Set to 1 for SHUTDOWN messages; 0 otherwise. **Note**: SHUTDOWN is not applicable to WebSphere Message Brokers.
reload	Set to 1 for RELOAD messages; 0 otherwise. **Note**: RELOAD is not applicable to WebSphere Message Brokers.

Example Calls to buildMessage

To build a message with application group “TestApp” and message type “TestMsgType”, call the buildMessage routine as follows:

```
buildMessage(&md, dataLength, data, "MQSTR",
   bufferLength, buffer, "TestApp", "testMsgType", 0.0);
```

Source Code for buildMessage

```
#include "MQSIrfheader.h"
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

int buildMessage(MQMD* md, long dataLength, char* data,
     char *dataFormat, int *bufferLength, char *buffer,
     char *applicationGroup, char *messageType,
     int shutdown, int reload)
{
     char optionBuffer[1024];
     int outputCursor = 0;
     int optionBufferLength = 0;
     MQMD tmpMd = {MQMD_DEFAULT};
     MQRFH header = {MQRFH_DEFAULT};

     memcpy(md, &tmpMd, sizeof(MQMD));
     memset (optionBuffer, 0, sizeof(optionBuffer));

     /*Construct the Option Buffer*/
     if (applicationGroup != NULL)
     {
          strcat(optionBuffer, "OPT_APP_GRP");
          strcat(optionBuffer, " ");
          strcat(optionBuffer, applicationGroup);
          strcat(optionBuffer, " ");
     }

     if (messageType != NULL)
     {
          strcat(optionBuffer, "OPT_MSG_TYPE");
          strcat(optionBuffer, " ");
          strcat(optionBuffer, messageType);
          strcat(optionBuffer, " ");
     }

     if (shutdown > 0)
```

```
        {
             strcat(optionBuffer, "OPT_SHUTDOWN");
             strcat(optionBuffer, " ");
             strcat(optionBuffer, "SHUTDOWN");
             strcat(optionBuffer, " ");
        }

        if (reload > 0)
        {
             strcat(optionBuffer, "OPT_RELOAD_RULE_SET");
             strcat(optionBuffer, " ");
             strcat(optionBuffer, "TRUE");
             strcat(optionBuffer, " ");
        }

        if (strlen(optionBuffer) > 0)
        {
             /*Remove Trailing Blank*/
             optionBufferLength = strlen(optionBuffer) - 1;
        }
        else
        {
             optionBufferLength = strlen(optionBuffer);
        }


        /*Construct the MQRFH structure*/
        header.StrucLength = sizeof(MQRFH) +
             optionBufferLength;

        if (dataFormat != NULL)
        {
             strncpy(header.Format, dataFormat,
                  sizeof(header.Format));
        }


   /*Make sure there is enough room in the buffer to hold*/
   /*the header, options and data*/
   if (*bufferLength <
       (sizeof(MQRFH) + optionBufferLength + dataLength))
   {
```

```
    return (0);
}

/*Transfer header, options, and data to the message */
/* buffer */
memcpy(buffer + outputCursor, &header, sizeof(MQRFH));
outputCursor += sizeof(MQRFH);
memcpy(buffer + outputCursor, optionBuffer,optionBufferLength);
outputCursor += optionBufferLength;
if (data != NULL)
{
     memcpy(buffer + outputCursor, data, dataLength);
     outputCursor += dataLength;
}
else
     {
       return(0);
     }

/*Return the size of the header + options + data*/
*bufferLength = outputCursor;

/*Set the message descriptor format field            */
/*to indicate that an MQIntegrator header is present */
/*in the message buffer.                             */
strncpy(md->Format, "MQRFH   ", sizeof(md->Format));

return(1);
}
```

Sample Application Using buildMessage

The following source code is from the amqsput0.c MQSeries sample application and is modified to use the buildMessage routine. The program functions the same as amqsput0, except it prepends an WebSphere Message Broker header to each message that is sends. The program assigns the application group “TestApp” and the message type “TestMsg” to each message that it puts.

```
/**************************************************************/
/*                                                            */
/* Program name: AMQSPUT0                                     */
/*                                                            */
/* Description: Sample C program that puts messages to        */
/* a message queue (example using MQPUT)                      */
/*                                                            */
/*  Statement:     Licensed Materials - Property of IBM       */
/*                                                            */
/*                 84H2000, 5765-B73                          */
/*                 84H2001, 6539-B42                          */
/*                 84H2002, 5765-B74                          */
/*                 84H2003, 5765-B75                          */
/*                 84H2004, 6539-B43                          */
/*                 (C) Copyright IBM Corp. 1994, 1997         */
/*                                                            */
/**************************************************************/
/*                                                            */
/* Function:                                                  */
/*                                                            */
/*                                                            */
/*   AMQSPUT0 is a sample C program to put messages on a message  */
/*   queue, and is an example of the use of MQPUT.            */
/*                                                            */
/*      -- messages are sent to the queue named by the parameter  */
/*                                                            */
/*      -- gets lines from StdIn, and adds each to target     */
/*         queue, taking each line of text as the content     */
```

```
/*         of a datagram message; the sample stops when a null    */
/*         line (or EOF) is read.                                 */
/*         New-line characters are removed.                       */
/*         If a line is longer than 99 characters it is broken up */
/*         into 99-character pieces. Each piece becomes the       */
/*         content of a datagram message.                         */
/*         If the length of a line is a multiple of 99 plus 1     */
/*         e.g. 199, the last piece will only contain a new-line  */
/*         character so will terminate the input.                 */
/*                                                                */
/*      -- writes a message for each MQI reason other than        */
/*         MQRC_NONE; stops if there is a MQI completion code     */
/*         of MQCC_FAILED                                         */
/*                                                                */
/*    Program logic:                                              */
/*         MQOPEN target queue for OUTPUT                         */
/*         while end of input file not reached,                   */
/*         .  read next line of text                              */
/*         .  MQPUT datagram message with text line as data       */
/*         MQCLOSE target queue                                   */
/*                                                                */
/*                                                                */
/****************************************************** *****/
/*                                                                */
/*   AMQSPUT0 has 2 parameters                                    */
/*                 - the name of the target queue (required)      */
/*                 - queue manager name (optional)                */
/*                                                                */
/******************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
   /* includes for MQI */
#include <cmqc.h>
int main(int argc, char **argv)
{
  /*  Declare file and character for sample input                 */
  FILE *fp;
  /*   Declare MQI structures needed                              */
  MQOD     od = {MQOD_DEFAULT};    /* Object Descriptor           */
  MQMD     md = {MQMD_DEFAULT};    /* Message Descriptor          */
  MQPMO   pmo = {MQPMO_DEFAULT};   /* put message options         */
```

```
  /** note, sample uses defaults where it can **/
MQHCONN  Hcon;                  /* connection handle           */
MQHOBJ   Hobj;                  /* object handle               */
MQLONG   O_options;             /* MQOPEN options              */
MQLONG   C_options;             /* MQCLOSE options             */
MQLONG   CompCode;              /* completion code             */
MQLONG   OpenCode;              /* MQOPEN completion code      */
MQLONG   Reason;                /* reason code                 */
MQLONG   CReason;               /* reason code for MQCONN      */
MQLONG   buflen;                /* buffer length               */
char     buffer[100];           /* message buffer              */
char     QMName[50];            /* queue manager name          */

/* buffer to hold MQIntegrator Header and Message data*/
char     newBuffer[1024];
/* size of new buffer */
int      newBufferLength = 1024;
printf("Sample AMQSPUT0 start\n");
if (argc < 2)
{
  printf("Required parameter missing - queue name\n");
  exit(99);
}

/*********************************************************************/
/*                                                                   */
/*   Connect to queue manager                                        */
/*                                                                   */
/*********************************************************************/
QMName[0] = 0;    /* default */
if (argc > 2)
  strcpy(QMName, argv[2]);
MQCONN(QMName,                  /* queue manager               */
       &Hcon,                   /* connection handle           */
       &CompCode,               /* completion code             */
       &CReason);               /* reason code                 */
/* report reason and stop if it failed     */
if (CompCode == MQCC_FAILED)
{
  printf("MQCONN ended with reason code %ld\n", CReason);
  exit( (int)CReason );
}
```

```
/*****************************************************************/
/*                                                               */
/*   Use parameter as the name of the target queue               */
/*                                                               */
/*****************************************************************/
strncpy(od.ObjectName, argv[1], (size_t)MQ_Q_NAME_LENGTH);
printf("target queue is %s\n", od.ObjectName);
/*****************************************************************/
/*                                                               */
/*   Open the target message queue for output                    */
/*                                                               */
/*****************************************************************/
O_options = MQOO_OUTPUT           /* open queue for output
    */
        + MQOO_FAIL_IF_QUIESCING; /* but not if MQM stopping     */
MQOPEN(Hcon,                      /* connection handle           */
       &od,                       /* object descriptor for queue */
       O_options,                 /* open options                */
       &Hobj,                     /* object handle               */
       &OpenCode,                 /* MQOPEN completion code      */
       &Reason);                  /* reason code                 */
/* report reason, if any; stop if failed      */
if (Reason != MQRC_NONE)
{
  printf("MQOPEN ended with reason code %ld\n", Reason);
}
if (OpenCode == MQCC_FAILED)
{
  printf("unable to open queue for output\n");
}
/*****************************************************************/
/*                                                               */
/*   Read lines from the file and put them to the message queue  */
/*   Loop until null line or end of file, or there is a failure  */
/*                                                               */
/*****************************************************************/
CompCode = OpenCode;        /* use MQOPEN result for initial test
    */
fp = stdin;
memcpy(md.Format,          /* character string format         */
       MQFMT_STRING, (size_t)MQ_FORMAT_LENGTH);
while (CompCode != MQCC_FAILED)
```

```
{
  if (fgets(buffer, sizeof(buffer), fp) != NULL)
  {
    buflen = strlen(buffer);       /* length without null        */
    if (buffer[buflen-1] == '\n')  /* last char is a new-line    */
    {
      buffer[buflen-1]  = '\0';   /* replace new-line with null  */
      --buflen;                   /* reduce buffer length        */
    }
  }
  else buflen = 0;               /* treat EOF same as null line  */
  /*************************************************************/
  /*                                                           */
  /*   Put each buffer to the message queue                    */
  /*                                                           */
  /*************************************************************/
  if (buflen > 0)
  {
    memcpy(md.MsgId,           /* reset MsgId to get a new one   */
           MQMI_NONE, sizeof(md.MsgId) );
    memcpy(md.CorrelId,        /* reset CorrelId to get a new one*/
           MQCI_NONE, sizeof(md.CorrelId) );

   buildMessage(&md, buflen, buffer, "MQSTR",
      &newBufferLength, newBuffer, "TestApp", "TestMsg",0, 0);
    MQPUT(Hcon,                /* connection handle              */
          Hobj,                /* object handle                  */
          &md,                 /* message descriptor             */
          &pmo,                /* default options (datagram)     */
         newBufferLength,  /* buffer length with MQIntegrator
     header   */
          newBuffer,  /* message buffer with MQIntegrator header */
               &CompCode,          /* completion code
    */
          &Reason);            /* reason code                    */
    /* report reason, if any */
    if (Reason != MQRC_NONE)
    {
      printf("MQPUT ended with reason code %ld\n", Reason);
    }
  }
  else   /* satisfy end condition when empty line is read */
```

```
    CompCode = MQCC_FAILED;
}
/******************************************************************/
/*                                                                */
/*   Close the target queue (if it was opened)                    */
/*                                                                */
/******************************************************************/
if (OpenCode != MQCC_FAILED)
{
  C_options = 0;                  /* no close options             */
  MQCLOSE(Hcon,                   /* connection handle            */
          &Hobj,                  /* object handle                */
          C_options,
          &CompCode,              /* completion code              */
          &Reason);               /* reason code                  */
                                  /* report reason, if any        */
  if (Reason != MQRC_NONE)
  {
    printf("MQCLOSE ended with reason code %ld\n", Reason);
  }
}
/******************************************************************/
/*                                                                */
/*   Disconnect from MQM if not already connected                 */
/*                                                                */
/******************************************************************/
if (CReason != MQRC_ALREADY_CONNECTED)
{
  MQDISC(&Hcon,                   /* connection handle            */
         &CompCode,               /* completion code              */
         &Reason);                /* reason code                  */
                                  /* report reason, if any        */
  if (Reason != MQRC_NONE)
  {
    printf("MQDISC ended with reason code %ld\n", Reason);
  }
}
/******************************************************************/
/*                                                                */
/* END OF AMQSPUT0                                                */
/*                                                                */
/******************************************************************/
```

```
    printf("Sample AMQSPUT0 end\n");
    return(0);                                              }
```

Appendix A
Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

```
IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.
```

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

```
IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan
```

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and / or changes in the product(s) and / or the program(s) described in this publication at any time without notice.

Any references in this document to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

```
IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England,
SO21 2JN.
```

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Any Performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments

may vary significantly. Some measurements may have been made on development-level systems and there is not guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information includes examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

this information contains sample application programs in source language, which illustrate programming techniques on varius operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

AIX	CICS	DB2
DB2 Universal Database	developerWorks	
Everyplace	FFST	First Failure Support Technology
IBM	IMS	IMS/ESA
iSeries	Language Environment	MXSeries
MVS	NetView	OS/400
OS/390	pSeries	RACF
RETAIN	RS/6000	SupportPac
Tivoli	VisualAge	WebSphere
xSeries	z/OS	zSeries

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries or both.

Pentium is a registered trademark of Intel.

UNIX is a registered trademark in the United States andother countries licensed exclusively through The Open Group.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

Index

O

P

R

S

T

Sending your comments to IBM
Rules and Formatter Extension for WebSphere Message Broker for Multi-platforms
System Management Guide

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book. Please limit your comments to the information in this book only and the way in which the information is presented.

To request additional publications or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

You can send your comments to IBM in any of the following ways:

- By mail:

  IBM United Kingdom Laboratories
  Hursley Park
  Winchester
  Hampshire
  SO21 2JN

- By fax:
  - From outside the U.K., use your international access code followed by 44 1962 870229
  - From within the U.K., use 01962 816151

Electronically, use the appropriate network ID:

- IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
- IBMLink: HURSLEY(IDRCF)

- Internet: idrcf@hursley.ibm.com

Whichever you use, ensure that you include:

- The publication number and title
- The page number or topic number to which your comment applies
- Your name / address / telephone number / fax number / network ID