

DB Mail™ 2.5

User's Guide

Supported database systems:

Oracle 7.3, 8.0, 8i, 9i, 10g

Microsoft SQL Server 6.5, 7, 2000, 2005

Sybase SQL Server and Sybase Adaptive
Server Enterprise 10.x, 11.x, 12.x

Sybase Adaptive Server Anywhere 6, 7, 8, 9

IBM DB2 UDB 5.x, 6.x, 7.x, 8.x for Unix, Linux and
Windows

Table of Contents

Table of Contents	3
About this guide	8
INTENDED AUDIENCE	8
CONVENTIONS USED IN THIS DOCUMENT	8
ABBREVIATIONS AND PRODUCT REFERENCE TERMS	9
TRADEMARKS	9
Introduction	10
CHAPTER 1, How DB Mail works	11
IMPLEMENTATION	11
Message processing workflow	11
Sample Scenarios	14
SUPPORTED MESSAGING METHODS AND PROTOCOLS	16
Emailing - sending electronic mail	16
Paging - sending numeric and text messages to pagers and cell phones (SMS)	16
Network messaging - sending interruptible network popup messages	17
System alerts - sending interruptible network alerts	17
Faxing - sending electronic faxes (paperless)	17
Voice Messaging- making phone calls and sending voice messages	18
SUPPORTED DATABASE SYSTEMS AND OPTIONS	18
Database Connectivity Requirements	19
Messaging features by DBMS	19
FREQUENTLY ASKED QUESTIONS (FAQ)	21
CHAPTER 2, Connecting To Your Database	23
CONNECTION METHODS AND REQUIREMENTS	23
PREPARING TO USE YOUR DATABASE	24
INSTALLING THE ODBC DRIVER OR NATIVE DATABASE DRIVER	24
DEFINING THE ODBC DATA SOURCE	24
TROUBLESHOOTING THE DATABASE CONNECTION	25
DATABASE PROFILES	25
CHAPTER 3, DB Mail database interfaces	26
ORACLE	26
MICROSOFT SQL SERVER	27
SYBASE SQL SERVER, ASE, ASA	28
Advanced version interface	28
Limited version interface	29
IBM DB2	32

Advanced version interface	32
Limited version interface.....	33
CHAPTER 4, Installation and Uninstallation	36
FRONT-END INSTALLATION	36
DB Mail Server Installation	36
VoMS Installation.....	37
BACK-END INSTALLATION	37
Requirements	37
Oracle	38
Microsoft SQL Server	40
Sybase SQL Server, ASE, ASA.....	41
IBM DB2.....	44
How to copy files (SQL Server example).....	46
How to FTP files (DB2 example)	46
Managing User Access to DB Mail features	47
TESTING.....	49
UNINSTALLATION.....	49
CHAPTER 5, Configuring DB Mail	51
CONFIGURING DATABASES OPTIONS	51
CONFIGURING EMAIL OPTIONS	54
CONFIGURING NEW MAPI PROFILE	56
CONFIGURING PAGER OPTIONS.....	59
CONFIGURING NETWORK POPUPS AND ALERTS OPTIONS.....	60
CONFIGURING FAX OPTIONS	61
CONFIGURING VOICE MESSAGING OPTIONS.....	63
CONFIGURING QUEUE OPTIONS.....	64
CONFIGURING ERROR-HANDLING OPTIONS	66
CONFIGURING SELF-HEALING AND MAINTENANCE OPTIONS	67
CONFIGURING ARCHIVING OPTIONS	69
CONFIGURING USER-ACCESS AND SECURITY.....	70
CHAPTER 6, Sending email messages	71
OVERVIEW	71
ORACLE	71
SEND_MAIL	71
ATTACH_FILE.....	74
ATTACH_DATA.....	78
CREATE_MAIL_FILE	80
DELETE_MAIL_FILE.....	82
MICROSOFT SQL SERVER.....	83

SendMail	83
AttachFile	86
AttachData	90
CreateMailFile	92
DeleteMailFile	93
SYBASE SQL SERVER, ASE, ASA	94
SendMail	94
AttachFile	97
AttachData	101
CreateMailFile	102
DeleteMailFile	105
IBM DB2	105
SendMail	105
AttachFile	109
AttachData	112
CreateMailFile	114
DeleteMailFile	116
CHAPTER 7, Sending SMS/pager messages	118
OVERVIEW	118
ORACLE	118
SEND_PAGE	118
MICROSOFT SQL SERVER	119
SendPage	119
SYBASE SQL SERVER, ASE, ASA	120
SendPage	120
IBM DB2	122
SendPage	122
CHAPTER 8, Sending network popup messages	125
OVERVIEW	125
ORACLE	125
SEND_POPUP_MESSAGE	125
MICROSOFT SQL SERVER	127
SendPopupMessage	127
SYBASE SQL SERVER, ASE, ASA	127
SendPopupMessage	127
IBM DB2	128
SendPopupMessage	128
CHAPTER 9, Sending system alerts	130

OVERVIEW	130
ORACLE	130
SEND_ALERT	130
MICROSOFT SQL SERVER	131
SendAlert.....	131
SYBASE SQL SERVER, ASE, ASA	132
SendAlert.....	132
IBM DB2.....	133
SendAlert.....	133
CHAPTER 10, Sending electronic faxes	134
OVERVIEW	134
CREATING AND MODIFYING COVER PAGES	134
ORACLE	136
SEND_FAX.....	136
SEND_FAX_EX.....	138
MICROSOFT SQL SERVER	141
SendFax	141
SendFaxEx.....	143
SYBASE SQL SERVER, ASE, ASA	147
SendFax	147
SendFaxEx.....	149
IBM DB2.....	153
SendFax	153
SendFaxEx.....	155
CHAPTER 11, Sending phone/voice messages	160
OVERVIEW	160
CREATING PRE-RECORDED SOUND MESSAGES AND MESSAGE SEGMENTS	160
CREATING DYNAMICALLY SYNTHESIZED VOICE MESSAGES AND MESSAGE SEGMENTS	161
ORACLE	162
SEND_VOICE	162
MICROSOFT SQL SERVER	166
SendVoice	166
SYBASE SQL SERVER, ASE, ASA	172
SendVoice	172
IBM DB2.....	176
SendVoice	176
CHAPTER 12, Helpful Tips and Recommendations	181
PERFORMANCE TIPS.....	181
USAGE TIPS	181

DATABASE PORTABILITY TIPS 182

CHAPTER 13, Troubleshooting and Maintenance 184

 BASIC TROUBLESHOOTING 184

 TROUBLESHOOTING MESSAGE PROCESSING 184

 TROUBLESHOOTING DATABASE OPERATIONS 184

 KNOWN ISSUES 184

APPENDIX A, Starting DB Mail on Computer Startup 186

APPENDIX B, Running DB Mail as a Windows NT service 187

APPENDIX C, Hardware and Software Requirements 188

APPENDIX D, Technical Support 190

APPENDIX E, Licensing 192

About this guide

This manual describes the features of the DB Mail product, including how to install and use the DB Mail graphical user interface and send out electronic communications such as Email, Fax, interruptible Network Popup and Alert messages and Pages using examples. The described features and how-to instructions apply to all the supported DBMS running on any platform, with sections describing the specifics of particular DBMS.

Intended audience

This document is for Database Administrators, Database Managers, System Administrators and Database Owners.

Conventions used in this document

This section describes the style conventions used in this document.

Italic

An *italic* font is used for filenames, URLs, emphasized text, and the first usage of technical terms.

Monospace

A monospaced font is used for code fragments and data elements.

Bold

A **bold** font is used for important messages, names of options, names of controls and menu items, and keys.

User Input

Keys are rendered in **bold** to stand out from other text. Key combinations that are meant to be typed simultaneously are rendered with "+" sign between the keys, such as:


Ctrl+F


Keys that are meant to be typed in sequence will be separated with commas, for example:

Alt+S, H

This would mean that the user is expected to type the Alt and S keys simultaneously and then to type the H key.

Graphical symbols

 This symbol is used to mark useful tips.

 This symbol is used to indicate important notes.

Abbreviations and product reference terms

DBMS – Database Management System

Oracle – This refers to all supported Oracle® database servers

SQL Server – This refers to all versions of Microsoft® SQL Server™ database servers.

ASE – This refers to all versions of the Sybase® SQL Server™ and Sybase® Adaptive Server® Enterprise database servers.

ASA – This refers to all versions of the Sybase® Adaptive Server® Anywhere database servers.

DB2 – This refers to all versions of the IBM® DB2® database servers.

Trademarks

DB Mail, 24x7 Automation Suite, 24x7 Scheduler, DB Tools for Oracle, VoMS are trademarks of SoftTree Technologies, Inc.

Windows 95, Windows 98, Windows NT, Windows 2000, Windows XP are registered trademarks of Microsoft Corporation. UNIX is the registered trademark of the X/Open Consortium. Sun, SunOS, Solaris, SPARC are trademarks or registered trademarks of Sun Microsystems, Inc. Ultrix, Digital UNIX and DEC are trademarks of Digital Equipment Corporation. HP-UX is a trademark of Hewlett-Packard Co. IRIX is a trademark of Silicon Graphics, Inc. AIX is a trademark of International Business Machines, Inc. AT&T is a trademark of American Telephone and Telegraph, Inc.

Microsoft SQL Server, Microsoft Outlook are registered trademarks of Microsoft Corporation.

Oracle is a registered trademark of Oracle Corporation.

Sybase, Adaptive Server, Adaptive Server Anywhere, Adaptive Server Enterprise, Anywhere Studio are registered trademarks of Sybase, Inc. or its subsidiaries.

IBM, DB2, UDB are registered trademarks of International Business Machines Corporation. Lotus Notes is a registered trademark of International Business Machines Corporation.

All other trademarks appearing in this document are trademarks of their respective owners. All rights reserved.

Introduction

E-business is fundamentally changing the way companies operate, going far beyond buying and selling over the Internet, or e-commerce. E-business enables businesses to leverage the use of technology to gain an advantage in the marketplace. With this technology, businesses can expand their customer base, provide faster, more efficient services, and develop more personalized services. In addition, the overhead associated with implementing business processes can be minimized or reduced by leveraging e-business technology to streamline interactions with customers, suppliers and business partners. Most modern e-business solutions are database-driven. Powerful relational databases systems play a central role in storing e-business data and automating e-business services. However, most commonly used database systems do not come inbuilt with all the features required for a successful e-business implementation. A lot of customization, tweaking and programming is required to fully integrate and automate these systems.

DB Mail provides an easy and efficient method for e-business enabling many database systems. With minimal effort, It allows you to automate such common e-business functions as sending electronic messages including e-mail and e-fax messages, pager and mobile phone messages, interruptible network messages and administrative alerts. And best of all, all these messages can be sent directly from your databases. Any application capable of connecting to your databases can also use DB Mail services with minimal programming.

DB Mail is also very easy to install, configure and use. Using DB Mail developers can easily add messaging functions to their homegrown applications. Database Administrators can automate alert functions for various database events and monitoring processes. For example, an application can send an alert whenever the inventory drops below a previously defined critical level, or an office management application can send an automated message to all users reminding them that it's now time to submit their attendance timesheets.

CHAPTER 1, How DB Mail works

Implementation

DB Mail is designed as a multi-threaded application that can be configured to simultaneously serve multiple databases servers. It provides a robust messaging gateway that allows internal and external database applications to easily create and send various messages including e-mails, e-faxes, network popup messages, pager and phone messages. DB Mail uses standard database communication network protocols, which allow it to support any Oracle, DB2, Sybase and Microsoft SQL Server database systems running on any platform.

DB Mail runs a separate daemon process for every configured database connection. In case one of the connected databases goes down, it does not stop other daemon processes connected to other databases. Each daemon process checks periodically for messages written to message pipe in the connected database and transfers new message to the central message queue. DB Mail also runs another asynchronous process called Message Queue Processor whose purpose is process messages in the central message queue. The Message Queue Processor verifies message data and delivers messages to the recipients. The actual delivery method varies for different message types.

Notes:

In all databases except Oracle, the DB Mail local message queue is table-based. All messages are written to the DBMAIL.PIPE table from where DB Mail database daemons transfer messages to the central message queue.

In Oracle databases DB Mail utilizes advanced Oracle messaging features based on the system DBMS_PIPE package. All messages are written to the virtual message pipe by calling functions available in DBMS_PIPE package. On the other end of the pipe DB Mail daemons constantly listen for new messages and wake up when new messages are written. From there, the processing is identical to the processing in other database systems.

See the next section for description of the complete message processing workflow.

Message processing workflow

Applications that use asynchronous operations (where more than a single operation can execute simultaneously) often have better availability, reliability, scalability, and usability than applications using only synchronous operations (where operations execute sequentially). DB Mail is designed with the goal of supporting asynchronous message processing, which allows database applications to write messages to a message queue and be immediately freed to serve other user requests.

After messages are written to the message queue, asynchronous DB Mail database daemon processes pick up queued messages and transfer them to the central message queue where messages are processed according to their priorities and timing.

Figure 1 shows the components of a message processing workflow diagram.

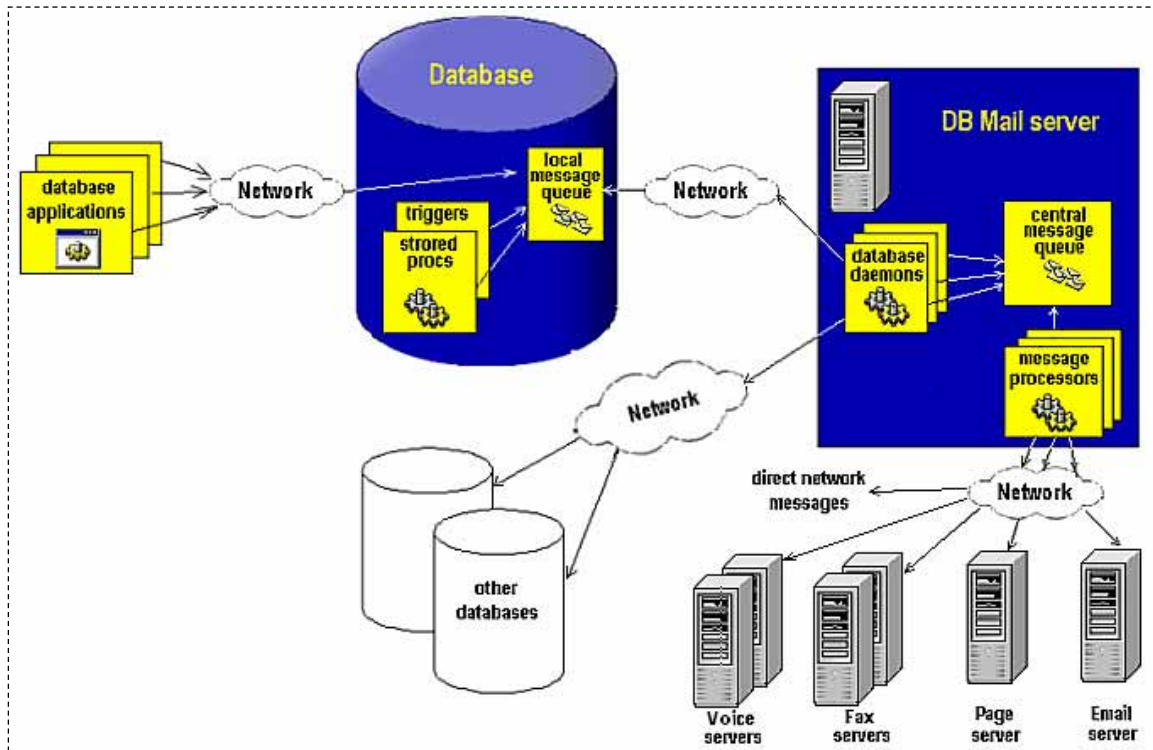


Figure 1: Message processing workflow

The components topics describe message workflow components.

Database applications

There are essentially 3 types of database applications:

- Internal applications that use stored code run entirely within the space of the database server. An example of internal application could be a standalone database stored procedure designed to periodically produce and then email some business report. Such procedures can be scheduled using one of the available database job scheduling methods.
- External applications have their code stored completely outside of the database. An example of such application could be a Visual Basic program (or other type of interactive or automatic program) that is run on a user's workstation. When executing, this program connects to the database, performs some data manipulations and then disconnects from the database.
- Mixed applications have some portions of their code stored and run in the database space while other parts of their code are stored and run outside the database. For example a Visual Basic application described in the previous paragraph could insert new records into an order entry table. The database portion of this application could be implemented as a database trigger for the order entry table, which is executed by the database in the event of new records being inserted into the table. Such trigger can perform some business data validation and email sales managers in case inventory level falls below a certain level. The same trigger could also create customer invoices for the inserted orders and automatically fax them to customers.

Database server

Database management systems containing local DB Mail queue and messaging interface procedures.

Local message queue

Database local message queue is used to temporarily store pending messages. In most systems database queue is implemented as a regular database table. Database applications normally write to the message queue using DB Mail interface functions and procedures.

Database daemons

Daemon processes run on the DB Mail server computer. Daemon processors monitor database-side message queues and transfer new messages to the DB Mail server.

Central message queue

Central message queue is the file based message queue where database daemons drop transferred messages.

Message processors

Message Processors are special processes that run on the DB Mail server computer. They are responsible for processing queued messages and delivering messages to specified recipients.

Email server

The email server such as Microsoft Exchange or Lotus Notes is used to send electronic mail. DB Mail server contains internal email client component, which it uses to connect and communicate to your corporate email server and send email messages. The connection and communication method differs for different email systems. DB Mail supports 3 most popular email interfaces: SMTP, Windows MAPI and Lotus Notes. You can use the DB Mail Server Console to choose which interface you want to use for sending email messages. It is recommended to use SMTP email interface whenever SMTP server is available. For more information about supported email interfaces and email options see [Configuring Email Options](#) and [Emailing - sending electronic mail](#) topics.


Page server

Page server is used to send SMS messages and alpha-numeric pages to cell phones and pagers using Simple Network Paging Protocol (SNPP). DB Mail server contains an internal SNPP client component, which is used to send SMS messages. It implements Simple Network Paging Protocol Version 2 as specified in RFC 1861. In order to deliver messages SNPP component connects and communicates to a SNPP server through the Internet. Most telephone service carriers provide SNPP servers to its users free of charge. Contact your cell phone service carrier to find out which SNPP server you can use. For more information about paging interface see [Configuring SMS and Pager Options](#) and [Paging - sending numeric and text messages to pagers and cell phones](#) topics.

Fax server


Fax server is used to transmit electronic faxes. DB Mail server contains internal file-to-FAX converter

and fax client components, which are used to print attached files to a Microsoft FAX print driver. The FAX driver converts files to fax-compatible TIFF images. The resulting TIFF images are then sent to a Windows 2000 or Windows XP computer connected to a fax-modem hooked to a phone line. Because all Windows 2000 and Windows XP computers feature built-in fax server software, any Windows 2000 or Windows XP computer can be used as a fax server. For more information about configuring fax interface see [Configuring Fax Options](#) and [Faxing - sending electronic faxes \(paperless\)](#) topics.

 Note: Fax transmissions are relatively slow operations. DB Mail provides built-in support for fax processing scaling out strategy (scaling out is the strategy that increases the capacity of an infrastructure tier to handle load by adding servers, thereby increasing the aggregate capacity of those servers). DB Mail server can be configured to work with multiple fax servers concurrently in order to increase the overall system throughput. As your fax processing volume grows you can add additional fax servers as needed and configure DB Mail server accordingly.

Voice server

Voice Message Server (VoMS) is used to make phone calls and send sound messages using either pre-recorded sound files, dynamically synthesized voice messages (using text-to-speech functions) or combination of both. DB Mail server contains internal VoMS client, which is used to send sound messages, text-to-speech, and mixed messages to a Windows NT, Windows 2000 or Windows XP computer connected to a standard voice-modem or Intel Dialogic phone board hooked to a phone line. The computer must be also running VoMS server software. A limited edition of VoMS is provided with each DB Mail license. For more information about configuring voice messaging interface see [Configuring Voice Messaging Options](#) and [Voice Messaging- making phone calls and sending voice messages](#) topics.

 Note: Phone calls and voice messaging are relatively slow operations. DB Mail provides built-in support for voice processing scaling out strategy (scaling out is the strategy that increases the capacity of an infrastructure tier to handle load by adding servers, thereby increasing the aggregate capacity of those servers). DB Mail server can be configured to work with multiple voice servers concurrently in order to increase the overall system throughput. As your call processing volume grows you can add additional voice servers as needed and configure DB Mail server accordingly.

Direct network messages

DB Mail currently supports two direct network messaging protocols and methods: sending network popup messages and sending administrative alerts. For more information about these methods and options see the following topics: [Configuring Network Poppups and Alerts Options](#), [Network messaging - sending interruptible network popup messages](#), [System alerts - sending interruptible network alerts](#) topics.

Sample Scenarios

On-line order processing application

A web-based Order Processing system accepts user orders and inserts every accepted order into an "order" table stored in the back-end Order Processing database. The INSERT operation fires a database trigger created for the INSERT operation on the "order" table. The trigger generates invoice data in a ready for faxing format and writes a message to the local message queue table. The database returns control back to the web portion of the application which then displays a confirmation message on the user's screen. This entire processing takes only a fraction of a second, as the application does not have to wait for the relatively slow fax operation to complete before getting the workflow control back and displaying the order confirmation.

On the other side of the local DB Mail queue, the asynchronous DB Mail daemon picks up the message

and transfers it to the central message queue where it is validated and processed by the DB Mail message processor. The DB Mail message processor then delivers the message.

Supply chain warehouse application

A Manufacturing system supplies finished goods to warehouses. Requests for finished goods may be fulfilled by the Manufacturer by supplying from internal stock or, if the required quantity is not available, by scheduling a production run. Since there could be a considerable time delay between receiving the order and informing the warehouse of shipment of goods, an asynchronous processing model is used. This allows a warehouse to proceed on other business, and allows the Manufacturer to callback to the Warehouse once the order has been fulfilled. In the event the order has been fulfilled the warehouse application calls DB Mail stored procedure that in turn writes callback message to the message queue.

As before, on the other side of the local DB Mail queue, the asynchronous DB Mail daemon picks up the message and transfers it to the central message queue where it is validated and processed by the DB Mail message processor. The DB Mail message processor then delivers the message.

Database administrative applications

An Oracle administrative application implemented as a database stored procedure runs periodically in an unattended mode on the database server. This procedure checks critical database metrics and invokes an administrative alert whenever a metric falls below certain threshold. Oracle's *DBMS_JOB* package is used to schedule and run the job. In order to send the alert the stored procedure calls the *SEND_ALERT* function from the *DB_MAIL* package. For critical conditions, the procedure also sends a pager message to the seniors DBAs by calling the *SEND_PAGE* function. Both the *SEND_ALERT* and *SEND_PAGE* functions write corresponding messages to the message queue. These messages are then picked up by DB Mail database daemon and get delivered to system administrators as interruptible administrative alerts that are displayed on the administrator's workstations and also as alphanumeric pager messages.

A Microsoft SQL Server application implemented as a database stored procedure runs periodically in an unattended mode in the MASTER database. This procedure checks all databases residing on the SQL Server instance for a LOG FULL condition. If such condition has been detected the procedure calls the *SEND_POPUP_MESSAGE* stored procedure which in turn writes new message to DB Mail message queue. This message is then picked up by DB Mail database daemon and gets broadcasted to all users notifying them that a database become unavailable and user applications accessing that database may become frozen. The message also tells users that their application will resume normal processing after database administrators truncate the filled log.

Appointment-reminder application

An automated application running every morning in a healthcare provider office checks in the local database for appointments scheduled on the same day. For every found appointment it then retrieves patient's name and phone number. It then calls DB Mail's *SEND_VOICE* function from the *DB_MAIL* package in order to send a pre-recorded message reminding the patient about the appointment time. The actual appointment time is pronounced using a computer generated voice. This entire processing takes only a few seconds, as the application does not have to wait for the relatively slow phone dialing operation to complete before getting the workflow control back.

On the other side of the local DB Mail queue, the asynchronous DB Mail daemon picks up all submitted messages and transfers them to the central message queue where they are validated and processed by the DB Mail message processor. The DB Mail message processor then communicates to the configured VoMS server which delivers the submitted messages to all patients.

Supported messaging methods and protocols.

Emailing - sending electronic mail

DB Mail supports several email protocols and interfaces, including:

- Windows Messaging Application Interface (MAPI)
- Simple Mail Transfer Protocol (SMTP)
- Lotus Notes (HTAPI)

All supported protocols allow sending plain text messages with and without attachments. In addition, DB Mail's SMTP protocol implementation supports sending rich-text format messages, HTML messages and XML messages. Such message formats provide greater control over email message appearance. They can also include in-line images, tables, formats and other graphical elements. The SMTP interface is also more robust and should be used whenever possible.

SMTP email interface does not require any additional software for sending email messages. You simply need to configure DB Mail to properly locate and authenticate to your SMTP email server.

MAPI email interface requires email client software such as Microsoft Outlook, Eudora, or Netscape Messenger installed on the computer running DB Mail Server. You do not need to have the email client program running in order to send email messages. The email client software just needs to be properly installed and configured.

Lotus Notes interface is mostly supported for legacy applications. Please use SMTP protocol if you are running Lotus Notes or Lotus Domino version 5 or later. In order to use Lotus Notes email interface Lotus Notes client software must be installed on the computer running DB Mail Server. You do not need to have the Lotus Notes client running in order to send email messages. The client software just needs to be properly installed and configured.

Tips:

Up to 255 external files or contents of database BLOB (Binary Large Object) columns can be attached to a single email message.

A single email message generated and sent via DB Mail can have multiple recipients specified in the *recipients* parameters of the *SEND_MAIL* function. DB Mail will automatically route such messages to all specified recipients.

A call to the *SEND_MAIL* function can be inserted in a regular SELECT statement with table columns or expressions specified for the function arguments. The database engine will invoke the *SEND_MAIL* function with different parameters as many times as the number of rows in the result set. See [CHAPTER 6. Sending email messages](#) for examples on using the *SEND_MAIL* function.

DB Mail SMTP interface supports different message content types such as *text/plain*, *text/html*, *text/xml*, etc.

Paging - sending numeric and text messages to pagers and cell phones (SMS)


DB Mail uses Simple Network Paging Protocol (SNPP) to send numeric and alphanumeric SMS messages through the Internet. DB Mail implements Simple Network Paging Protocol Version 2 as specified in RFC 1861. In the case of a numeric message the number of digits received by the recipients is usually limited to 8 to 12. In the case of an alphanumeric message, the number of characters received depends on the phone service carrier and is usually limited to 250 characters. Keeping your message short and concise helps to ensure that the recipient gets the most out of your message.

In order to allow sending numeric and text messages you need to configure DB Mail so it can locate and authenticate to your SNPP email server. Contact your mobile pager/cell phone carrier or your local phone company to find out your SNPP server address.

Network messaging - sending interruptible network popup messages

Windows network messaging allows sending arbitrary messages to a registered message alias such as network computer name or user name. The message appears on the destination computer as an interruptible popup message box. The user is required to click the OK button displayed on the message box in order to close the message and continue working. Because the size of the message box is somewhat limited to the computer screen, brief network messages are usually used to send important notifications or alerts.


DB Mail uses NetMessage protocol supported on Windows NT/2000/XP/2003 systems. Users running Windows 95/98/Me systems are unable to receive network messages.

 **Important Note:** The *Messenger* service must be running on your computer if you want to receive network messages. By default *Messenger* service is installed and turned on all NT/2000/XP/2003 systems. If the service is not running use Services applet in Windows Control Panel to change service start type to *Automatic* and start that service.

System alerts - sending interruptible network alerts

Sending interruptible system alerts is similar to sending network popup messages described in the previous topic. The only real difference is that network popup messages are sent to named recipients while administrative alerts are sent to a group of alert subscribers, which usually includes network administrators and other technical personal. Obviously the second method is more appropriate for sending administrative messages.

DB Mail alert functions rely on Windows NT LAN Manager protocol and more specifically on Windows NT *Alerter* service available on NT/2000/XP/2003 systems. Users running Windows 95/98/Me systems are unable to receive administrative alerts.

 **Important Note:** The *Alerter* service must be running if you want to send administrative alerts when something goes wrong on the server. The *Alerter* service also requires the *Messenger* service to be running.


Faxing - sending electronic faxes (paperless)

DB Mail supports sending outbound electronic faxes. DB Mail uses Microsoft Windows © Fax API available in Windows 2000 and later to create and transmit faxes. Below is the description of this processing workflow.

You use DB Mail stored procedures in your database to dynamically create text, HTML or XML documents and save them in the DB Mail queue table or as external files. DB Mail then opens each document and converts it to a FAX-compatible TIFF image. After this conversion takes place, a cover page is optionally added and the fax is handed off to the digital modem. The modem dials the recipient fax telephone number, listens for a fax signal, and sends the fax to the recipient's fax machine. After the fax has been sent successfully, DB Mail optionally sends a confirmation to the sender via e-mail. If the fax transmission fails after three or more attempts, a failure notification is optionally sent to the sender via e-mail. This failure notification indicates whether the fax failed because of a busy signal, no

answer, or the telephone number dialed was not a fax number.

The actual modem and fax transmission can be performed on the computer running DB Mail Server or any other Windows 2000 or later computer connected to the same network and featuring internal or external Class II digital fax-modem. Multi-modem digital adapters can be also used for high performance fax processing. Such multi-modem adapters are capable of sending several fax messages simultaneously.


 **Tip:** DB Mail implements advanced message queuing which allows true high-performance asynchronous database processing, as the performance is not affected by the performance of the fax processing. Database applications simply call DB Mail interface functions that quickly write messages to the message queue and immediately free applications making it unnecessary for the applications to wait for the slower fax processing.

Voice Messaging - making phone calls and sending voice messages

DB Mail supports sending pre-recorded and dynamically generated voice messages using advanced text-to-speech technologies. For dynamic text-to-speech messages DB Mail uses Microsoft Windows ® Speech API available in Windows 2000 and later to generate dynamic sound files. It then uses the VoMS (SoftTree Technologies' Voice Message Server™) software to deliver the messages over a regular phone line as an automated phone call. Below is the description of this processing workflow.

You use DB Mail stored procedures in your database to send voice messages. Messages can contain references to pre-recorded sound files or simply contain text that you want to say upon successful phone call. Messages can be also assembled from multiple parts containing both sound files and text segments. For every sent message DB Mail message queue processor picks up the message description, voice and text parts and submits them to the VoMS server that can be running on the same or different computer on your network. The VoMS server in turn converts all text segments to intermediate sound files and merges all referenced sound files into one single message. The resulting message is then handed off to the VoMS message processing engine that uses either a local modem or Intel Dialogic board to dial the recipient's phone number, listen for a human voice answer, and then speak the message. If the processing fails, for example, there is no modem or phone line available or answering human voice has not been detected after three or more attempts, a failure notification is optionally sent to the message sender via e-mail. This failure notification indicates whether the phone call failed because of a busy signal, no answer, call hang-up before completion, or the dialed number is not in service.

The actual phone call can be made from the computer running DB Mail Server or any other Windows 2000 or later computer connected to the same network and running VoMS software. The computer must have internal or external voice modem or Intel Dialog board.

 **Tip:** DB Mail implements advanced message queuing which allows true high-performance asynchronous database processing, as the performance is not affected by the performance of the voice call processing. Database applications simply call DB Mail interface functions that quickly write messages to the message queue and immediately free applications making it unnecessary for the applications to wait for the slower voice call processing.

Supported database systems and options

DB Mail supports a number of widely used database management systems (DBMS). Among them are Oracle 7, Oracle 8/8i, Oracle 9 database servers, IBM DB2 database server version 6.1 and later, Microsoft SQL Servers version 6.5 and later, Sybase SQL Server and Adaptive Server Enterprise (ASE) versions 10.0 and later, Sybase Adaptive Server Anywhere version 6.0 and later.

These DBMS can be installed and running on a variety of systems including but not limited to UNIX, Linux, Netware, OS2, OS390, OS400, Windows NT/2000/XP. DB Mail can work with all of them provided a database connection can be made from DB Mail server computer to a database server. A single DB Mail server can be configured to connect to and process messages from multiple heterogeneous database systems simultaneously.


Database Connectivity Requirements

DB Mail supports two database connection methods – connections using standard ODBC interface and connections using native database drivers. DB Mail software is shipped with native database drivers for Oracle OCI, Sybase CT-Lib and Microsoft SQL Server DB-Lib.


The following table describes supported database interfaces and connection methods.

DBMS Name	Oracle	Microsoft SQL Server	IBM DB2	Sybase SQL Server, Sybase ASE; Sybase ASA
Connection Method				
ODBC	X	X	X	X
Native database drivers	X ¹	X ²		X ³


X¹ – Oracle native driver connection requires Oracle client software installed on DB Mail computer. DB Mail native drivers for Oracle can work with Oracle SQL*Net v2, Net8 and later. Oracle OCI must be installed as a part of the client.

 **Tip:** You can verify Oracle client installation using Oracle SQL*Plus utility. As long as you can connect from SQL*Plus you should be able to connect from DB Mail.

X² – Microsoft SQL Server native driver connection requires SQL Server client software installed on DB Mail computer. The client software must include DB-Lib files.

 **Tip:** You can verify SQL Server client installation using Microsoft ISQL utility or Query Analyzer (available in SQL Server 7 and later). As long as you can connect from these utilities you should be able to connect from DB Mail.

X³ - Sybase SQL Server, ASE and ASA native driver connection requires Sybase client software installed on DB Mail computer. The client software must include CT-Lib files.

 **Tip:** You can verify Sybase client installation using Sybase ISQL utility. As long as you can connect from ISQL you should be able to connect from DB Mail.

Messaging features by DBMS

The following table describes DB Mail features supported in different database systems. Note that by "single email message" we mean a single DB Mail message, which can have one or more recipients.

DBMS Name and Version	Oracle				Microsoft SQL Server 6.5, 7, 2000 and later		IBM DB2 6.1, 6.2, 7.x, 8.x and later		Sybase SQL Server, Sybase ASE 10.x, 11.x, 12.x and later; Sybase ASA 6.x and later	
	7.3	8.0	8i	9i and later	6.5	7, 2000 and later	With Java support	Without Java support	With Java support	Without Java support
Send single email, page, alert or popup message using single SQL or procedural statement	X	X	X	X	X	X	X	X	X	X
Send multiple email, page, alert or popup messages using single SQL or procedural statement (such as SELECT from table which returns a multi-row result set)	X	X	X	X	X ¹	X ¹	X ¹	X ¹	X ¹	X ¹
Send single email message with attachments stored inside database using multiple procedural statements (at least 2 procedural statements required for every message)		X	X	X	X	X	X	X	X	
Send multiple email messages with attachments stored inside database using single SQL or procedural statement (such as SELECT from table which returns a multi-row result set)		X	X	X		X ²	X ³		X	
Send single email message with attachments stored outside database using multiple procedural statements (at least 2 procedural statements required for every message)		X	X	X	X	X	X			
Send multiple email messages with attachments stored outside database using single SQL or procedural statement (such as SELECT from table which returns a multi-row result set)		X	X	X		X ²	X ³			
Send single fax message using multiple SQL or procedural statement		X	X	X	X	X	X		X	

Send multiple fax messages using single SQL or procedural statement (such as SELECT from table which returns a multi-row result set)		X	X	X		X ²	X ³			
Send single voice message (text-to-speech) using multiple SQL or procedural statement	X	X	X	X	X	X	X	X	X	X
Send multiple voice messages (text-to-speech) using single SQL or procedural statement (such as SELECT from table which returns a multi-row result set)		X	X	X		X ²	X ³			
Send single voice message (pre-recorded sound files) using multiple SQL or procedural statement	X	X	X	X	X	X		X		X
Send multiple voice messages (pre-recorded sound files) using single SQL or procedural statement (such as SELECT from table which returns a multi-row result set)		X	X	X		X ²	X ³			
Message Archival	X	X	X	X	X	X	X	X	X	X
Message Logging and Auditing	X	X	X	X	X	X	X	X	X	X

X¹ - Can be implemented using single INSERT INTO ... SELECT statement.

X² - Can be implemented using user-defined scalar SQL FUNCTION which would call DB Mail stored procedures passing through function parameters.

X³ - Can be implemented using user-defined scalar SQL FUNCTION which would call DB Mail stored procedure passing through function parameters. DB2 database must be configured to support user-defined SQL functions. C/C++ compiler must be installed and configured on the database server.

Frequently Asked Questions (FAQ)

Q: Do I need to have a dedicated server for DB Mail? Can I install it on my database server?

A: No, you do not need a dedicated server. Yes, you can install DB Mail server on the computer running your database server only if that computer uses Windows 2000 and later as an operation system. In general you can install DB Mail on any computer in your organization as long as that computer can access your database servers via a regular TCP/IP network connection.

Q: Do I need an email server or client software installed on the machine running DB Mail? On my database servers?

A: If you choose to use SMTP email interface the answer is No. If you choose to use MAPI or Lotus

Notes email interface you will need an email client software (such as Microsoft Outlook, Lotus Notes, Netscape Messenger, Eudora or similar) installed on DB Mail computer.

Q: Do I need any special hardware to run DB Mail?

A: No. DB Mail software runs on any computer hosting Windows 2000, Windows XP and later.

CHAPTER 2, Connecting To Your Database

DB Mail can connect to a database using either an ODBC interface or a native database driver. DB Mail software includes native database drivers for Oracle 7, 8, 9 and 10, Sybase SQL Server and Adaptive Server Enterprise, and Microsoft SQL Server. DB Mail currently does not include native drivers for DB2 connections. You must use an ODBC connection to connect to DB2 database systems. ODBC drivers for DB2 and ASA are available from IBM, Sybase and other vendors.

Connection methods and requirements

Figure 2 illustrates required components for different database connection methods.

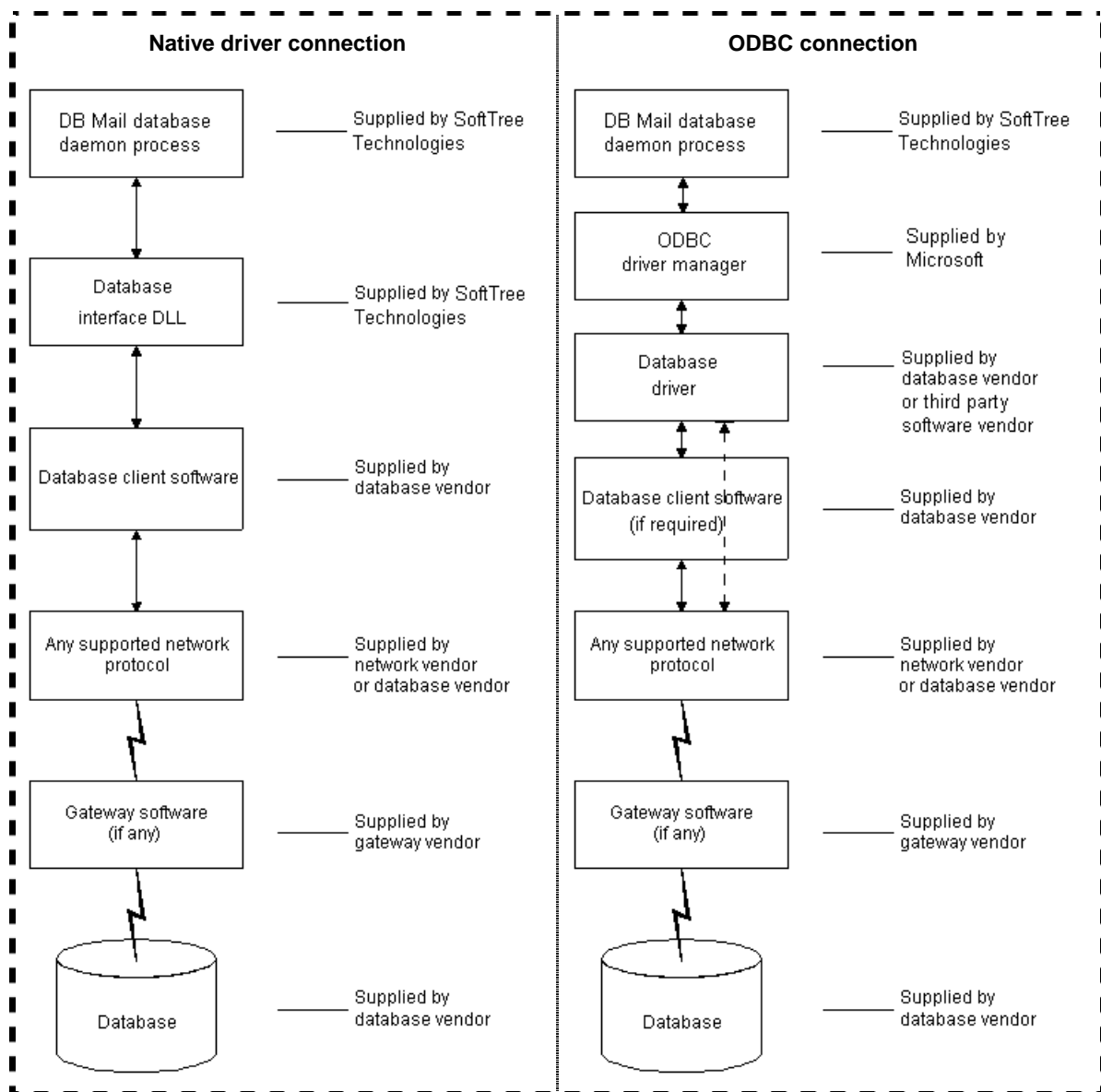


Figure 1: Connection methods diagram

Preparing to use your database

Preparing the database ensures that you will be able to access and use your data. The requirements differ for each database but, in general, preparing a database involves the following steps:

1. If network software is required, make sure it is properly installed and configured at your site and on the client machine.
2. Make sure the required database server software is properly installed and configured.
3. Make sure the required database client software is properly installed and configured on the DB Mail computer. (Typically, DB Mail is installed on a Windows NT workstation or server)



Important Note: You must install the appropriate client software for your database server and operating system platform before you can connect to the database. See your database vendor for specific information on where to obtain and how to install the client software.

Installing the ODBC driver or native database driver

To connect DB Mail to your database, you must install the ODBC driver or native database driver that accesses the database. Select the desired driver or database interface when prompted by the Setup program.

Defining the ODBC data source

Data that you access through an ODBC driver is referred to as an ODBC data source. An ODBC data source consists of the data and associated DBMS or file manager, operating system, and (if present) network software. When you define an ODBC data source, you provide information about the data source that the driver needs for the connection. Defining an ODBC data source is also referred to as configuring the data source. You can use the standard Windows ODBC Manager software to create and modify ODBC data sources. To start the ODBC Manager you will need to do the following:

From Windows Control Panel

1. Click the Windows **Start** button.
2. Select **Settings** menu, then select **Control Panel**. The Control Panel window will appear.
3. Double-click the **Administrative Tools** icon.
4. Double-click the **Data Sources (ODBC)** icon.

Completing the ODBC setup dialog box

Define an ODBC data source by completing the ODBC setup dialog box for the ODBC driver you have previously selected to access the data source. The content and layout of the ODBC setup dialog box

will vary for each driver, but most ODBC setup dialog boxes require you to supply the following information:

- Data source name and location,
- Data source description (optional),
- Other DBMS-specific connection parameters.

After you have created a data source, you can use it in the database profile that you create in the Database Profile option. Refer to the [Database Profiles](#) topic for details.

Troubleshooting the database connection

DB Mail supports two methods for tracing database connections in order to troubleshoot problems:

- **Database Trace** - The Database Trace tool records the internal commands that DB Mail executes while communicating with a database. Database Trace writes its output into a text file named PBTRACE.LOG, which is created in the Windows home directory. You can view the contents of the log file using any text editor. To enable database tracing, type "TRACE " (without quotes) in front of the chosen driver name in the [Configuring Databases Options](#).
- **ODBC Driver Manager Trace** - The ODBC Driver Manager Trace tool records information about the ODBC API calls made by DB Mail while connected to an ODBC data source. The ODBC Driver Manager Trace writes its output into a file named SQL.LOG (by default) located in the Windows home directory or to a log file that you specify. You can view the ODBC Driver Manager Trace log at any time by using any text editor.

Database Profiles

DB Mail uses database profiles as a way to simplify database connections and to provide naming methods. A **database profile** is a named set of parameters stored in the system registry under the DB Mail key. The profile data contains both the connection parameters for a particular database system and DB Mail configuration parameters related to that system, for example database server name, database user name and so on. Database profiles can be configured using the methods described in the later section titled **Configuring Database Options**.

CHAPTER 3, DB Mail database interfaces

This section describes the various Database interfaces supported by DB Mail. You will need to be aware of specific functionality provided by the DBMS of your choice that DB Mail uses.

Oracle

The Oracle messaging interface is implemented as a single PL/SQL package called DB_MAIL that encapsulates all the messaging calls. This package is located in the SYSTEM schema. DB Mail supports many functions and options in the more recent Oracle versions.

The DB_MAIL package encapsulates the following functions that can be called from SQL or PL/SQL code.

Function Name	Description
SEND_MAIL	Sends Email message, including optional email attachments
ATTACH_DATA ¹	Loads the MAIL_ATTACH table with BLOB values (Binary Large Objects such as pictures, files and other binary objects that are stored in database tables)
ATTACH_FILE ¹	Loads contents of external files (BFILES) to the MAIL_ATTACH table
SEND_FAX ¹	Sends faxes
SEND_FAX_EX ¹	Sends advanced faxes with complete control over cover pages including fill-in sender's contact information, messages on the cover page as well as requests automatic fax status notifications by email
SEND_PAGE	Sends alpha-numeric page messages (also called SMS messages) using SNPP protocol to pagers and cell phones
SEND_POPUP_MESSAGE	Sends network pop-up messages to specified logged in users
SEND_ALERT	Sends network pop-up alerts to system administrators and other personnel whose names are referenced in the Alert service
SEND_VOICE	Makes automated phone calls, performs call progress detection using intelligent voice recognition engine and then speaks either pre-recorded or dynamically synthesized sound messages
CREATE_MAIL_FILE ¹	Auxiliary procedure to write text, HTML and other flat text files on the server and then fax or email them as attachments. Files can be created only in the directory specified in the MAILSTORE directory object.

DELETE_MAIL_FILE ¹	Deletes files created by the CREATE_MAIL_FILE procedure
-------------------------------	---

¹ -- This function is not available in Oracle 7.3

For a complete description of each procedure and code examples see the following chapters:

[CHAPTER 6. Sending email messages](#)

[CHAPTER 7. Sending SMS/pager messages](#)

[CHAPTER 8. Sending network popup messages](#)

[CHAPTER 9. Sending system alerts](#)

[CHAPTER 10. Sending electronic faxes](#)

[CHAPTER 11. Sending phone/voice messages](#)

Microsoft SQL Server

The SQL Server messaging interface is implemented as a set of Transact-SQL and extended stored procedures. These procedures are located in the DBMAIL schema in the MASTER database. The following procedures can be called from SQL or Transact-SQL code or from other database applications.

Procedure Name	Description
SendMail	Sends Email message, including optional email attachments
AttachData	Loads the ATTACH table with BLOB values (Binary Large Objects such as pictures, files and other binary objects that are stored in database tables)
AttachFile	Loads contents of external files to the ATTACH table
SendFax	Sends faxes
SendFaxEx	Sends advanced faxes with complete control over cover pages including fill-in sender's contact information, messages on the cover page as well as requests automatic fax status notifications by email
SendPage	Sends alpha-numeric page messages (also called SMS messages) using SNPP protocol to pagers and cell phones
SendPopupMessage	Sends network pop-up messages to specified logged in users
SendAlert	Sends network pop-up alerts to system administrators and other personnel whose names are referenced in the Alert service
SendVoice	Makes automated phone calls, performs call progress detection using intelligent voice recognition engine and

	then speaks either pre-recorded or dynamically synthesized sound messages
CreateMailFile	Auxiliary procedure to write text, HTML and other flat text files on the server and then fax or email them as attachments
DeleteMailFile	Deletes files created by the CreateMailFile procedure

For complete description of each procedure and code examples see the following chapters:

[CHAPTER 6. Sending email messages](#)

[CHAPTER 7. Sending SMS/pager messages](#)

[CHAPTER 8. Sending network popup messages](#)

[CHAPTER 9. Sending system alerts](#)

[CHAPTER 10. Sending electronic faxes](#)

[CHAPTER 11. Sending phone/voice messages](#)

Sybase SQL Server, ASE, ASA

Two different versions of DB Mail database interface can be installed in Sybase databases:

- **Advanced version** – this version can be installed if your Sybase database supports Java stored procedures.
- **Limited version** – this version can be installed on any Sybase database as it does not require Java support in the database. However, DB Mail will be limited to sending simple message types only and will not support email messages with attachments as well as electronic faxes.

Advanced version interface

This messaging interface is implemented as a set of Java stored procedures. These procedures are located in the DBMAIL schema in the SYBSYSTEMPROCS database. The following procedures can be called from SQL or Transact-SQL code or from other database applications.

Procedure Name	Description
SendMail	Sends Email message, including optional email attachments
AttachData	Loads the ATTACH table with BLOB values (Binary Large Objects such as pictures, files and other binary objects that are stored in database tables)
AttachFile	Merges text file pieces previously inserted into the MAIL_FILE table into a single text BLOB and then loads the resulting BLOB into the ATTACH table as a named file.

SendFax	Sends faxes
SendFaxEx	Sends advanced faxes with complete control over cover pages including fill-in sender's contact information, messages on the cover page as well as request automatic fax status notifications by email
SendPage	Sends alpha-numeric page messages (also called SMS messages) using SNPP protocol to pagers and cell phones
SendPopupMessage	Sends network pop-up messages to specified logged in users
SendAlert	Sends network pop-up alerts to system administrators and other personnel whose names are referenced in the Alert service
SendVoice	Makes automated phone calls, performs call progress detection using intelligent voice recognition engine and then speaks either pre-recorded or dynamically synthesized sound messages
CreateMailFile	Auxiliary procedure to write text, HTML and other flat text files on the server and then fax or email them as attachments. Because Sybase does not currently support accessing external files from Java stored procedures, the contents of the files is saved in 255 character long pieces in the MAIL_FILE table. Use AttachFile procedure to merge these pieces together and load them as a single BLOB into the ATTACH table.
DeleteMailFile	Deletes file from the MAIL_FILE table.

For complete description of each procedure and code examples see the following chapters:

[CHAPTER 6. Sending email messages](#)

[CHAPTER 7. Sending SMS/pager messages](#)

[CHAPTER 8. Sending network popup messages](#)

[CHAPTER 9. Sending system alerts](#)

[CHAPTER 10. Sending electronic faxes](#)

[CHAPTER 11. Sending phone/voice messages](#)

Limited version interface

This messaging interface is limited to direct INSERTs into DBMAIL.PIPE table located in SYBSYSTEMPROCS database. For different message types you should use different columns of this table.

Creating email messages

The following columns in the DBMAIL.PIPE table can be used to create email messages.

Column Name	Description
MESSAGE_TYPE	A VARCHAR column whose value must be <i>EMAIL</i> for email message types.
RECIPIENTS	A VARCHAR column whose value is the email recipient address to send the message to. If you want to send the same message to multiple recipients you can use comma-separated list of email recipient addresses instead of sending multiple messages.
SUBJECT	A VARCHAR column whose value is the email subject.
MESSAGE	A VARCHAR column whose value is the actual email message. The format of the text in the message should match the value inserted into CONTENT_TYPE column.
REPLY_TO	A VARCHAR column whose value is the email address of the sender.
CONTENT_TYPE	A VARCHAR column whose value describes the format of the email message. You can use standard Internet formats. For example, for plain text email messages specify <i>text/plain</i> , for rich text messages specify <i>text/rtf</i> , for HTML email messages specify <i>text/html</i> , for XML email messages specify <i>text/xml</i> .

Example:

```
INSERT INTO dbmail.pipe (message_type, recipients, subject, message,
reply_to, content_type)
VALUES ('EMAIL', 'customer@custcompany.com', 'Test message', 'Test message,
please ignore.', 'me@mycompany.com', 'text/plain');
```

Creating alphanumeric pager and phone messages

The following columns in the DBMAIL.PIPE table can be used to create page messages.

Column Name	Description
MESSAGE_TYPE	A VARCHAR column whose value must be <i>PAGE</i> for page message types.
RECIPIENTS	A VARCHAR column whose value is the pager or cell phone number to send the message to. If you want to send the same message to multiple recipients you can use comma-separated list of numbers instead of sending multiple messages. The number format must conform to the standard used by your SNPP service provider.
MESSAGE	A VARCHAR column whose value is the actual message.

Example:

```
INSERT INTO dbmail.pipe (message_type, recipient, message)
VALUES ('PAGE', '1234567890', 'Test message, please ignore.');
```

Creating administrative alerts

The following columns in the DBMAIL.PIPE table can be used to create administrative alerts.

Column Name	Description
MESSAGE_TYPE	A VARCHAR column whose value must be <i>ALERT</i> for alert message types.
MESSAGE	A VARCHAR column whose value is the actual message.

Example:

```
INSERT INTO dbmail.pipe (message_type, message)
VALUES ('ALERT', 'Test system alert message, please ignore.');
```

Creating network popup messages

The following columns in the DBMAIL.PIPE table can be used to create popup messages.

Column Name	Description
MESSAGE_TYPE	A VARCHAR column whose value must be <i>NET</i> for network popup message types.
RECIPIENTS	A VARCHAR column whose value is the network username, network computer name, or network messaging name to send the message to. If you want to send the same message to multiple recipients you can use comma-separated list of recipient names instead of sending multiple messages. If you do not specify RECIPIENTS value or insert NULL value into column, DB Mail will broadcast the message to all the names in your domain or workgroup.
MESSAGE	A VARCHAR column whose value is the actual message.

Example:

```
INSERT INTO dbmail.pipe (message_type, recipient, message)
VALUES ('NET', 'MY COMPUTER NAME' 'Test message, please ignore.');
```

IBM DB2

Two different versions of DB Mail database interface can be installed in DB2 databases:

- **Advanced version** – this version can be installed if your DB2 database supports Java stored procedures.
- **Limited version** – this version can be installed on any DB2 database as it does not require Java support in the database. However, DB Mail will be limited to sending simple message types only and will not support email messages with attachments as well as electronic faxes.

Advanced version interface

This messaging interface is implemented as a set of Java stored procedures. These procedures are located in the DBMAIL schema. The following procedures can be called from SQL and compound SQL statements as well as from other database applications that are capable of calling DB2 stored procedures.

Procedure Name	Description
SendMail	Sends Email message, including optional email attachments
AttachData	Loads the ATTACH table with BLOB values (Binary Large Objects such as pictures, files and other binary objects that are stored in database tables)
AttachFile	Loads contents of external files to the ATTACH table
SendFax	Sends faxes
SendFaxEx	Sends advanced faxes with complete control over cover pages including fill-in sender's contact information, messages on the cover page as well as request automatic fax status notifications by email
SendPage	Sends alpha-numeric page messages (also called SMS messages) using SNPP protocol to pagers and cell phones
SendPopupMessage	Sends network pop-up messages to specified logged in users
SendAlert	Sends network pop-up alerts to system administrators and other personnel whose names are referenced in the Alert service
SendVoice	Makes automated phone calls, performs call progress detection using intelligent voice recognition engine and then speaks either pre-recorded or dynamically synthesized sound messages
CreateMailFile	Auxiliary procedure to write text, HTML and other flat text files on the server and then fax or email them as attachments

DeleteMailFile	Deletes files created by the CreateMailFile procedure
----------------	---

For complete description of each procedure and code examples see the following chapters:

[CHAPTER 6. Sending email messages](#)

[CHAPTER 7. Sending SMS/pager messages](#)

[CHAPTER 8. Sending network popup messages](#)

[CHAPTER 9. Sending system alerts](#)

[CHAPTER 10. Sending electronic faxes](#)

[CHAPTER 11. Sending phone/voice messages](#)

Limited version interface

This messaging interface is limited to direct INSERTs into DBMAIL.PIPE table. For different message types you should use different columns of this table.

Creating email messages

The following columns in the DBMAIL.PIPE table can be used to create email messages.

Column Name	Description
MESSAGE_TYPE	A VARCHAR column whose value must be <i>EMAIL</i> for email message types.
RECIPIENTS	A VARCHAR column whose value is the email recipient address to send the message to. If you want to send the same message to multiple recipients you can use comma-separated list of email recipient addresses instead of sending multiple messages.
SUBJECT	A VARCHAR column whose value is the email subject.
MESSAGE	A VARCHAR column whose value is the actual email message. The format of the text in the message should match the value inserted into CONTENT_TYPE column.
REPLY_TO	A VARCHAR column whose value is the email address of the sender.
CONTENT_TYPE	A VARCHAR column whose value describes format of the email message. You can use standard Internet formats. For example, for plain text email messages specify <i>text/plain</i> , for rich text messages specify <i>text/rtf</i> , for HTML email messages specify <i>text/html</i> , for XML email messages specify <i>text/xml</i> .

Example:

```
INSERT INTO dbmail.pipe (message_type, recipients, subject, message,
reply_to, content_type)
```

```
VALUES ('EMAIL', 'customer@custcompany.com', 'Test message', 'Test message,
please ignore.', 'me@mycompany.com', 'text/plain');
```

Creating alphanumeric pager and phone messages

The following columns in the DBMAIL.PIPE table can be used to create page messages.

Column Name	Description
MESSAGE_TYPE	A VARCHAR column whose value must be <i>PAGE</i> for page message types.
RECIPIENTS	A VARCHAR column whose value is the pager or cell phone number to send the message to. If you want to send the same message to multiple recipients you can use comma-separated list of numbers instead of sending multiple messages. The number format must conform to the standard used by your SNPP service provider.
MESSAGE	A VARCHAR column whose value is the actual message.

Example:

```
INSERT INTO dbmail.pipe (message_type, recipient, message)
VALUES ('PAGE', '1234567890', 'Test message, please ignore.');
```

Creating administrative alerts

The following columns in the DBMAIL.PIPE table can be used to create administrative alerts.

Column Name	Description
MESSAGE_TYPE	A VARCHAR column whose value must be <i>ALERT</i> for alert message types.
MESSAGE	A VARCHAR column whose value is the actual message.

Example:

```
INSERT INTO dbmail.pipe (message_type, message)
VALUES ('ALERT', 'Test system alert message, please ignore.');
```

Creating network popup messages

The following columns in the DBMAIL.PIPE table can be used to create popup messages.

Column Name	Description
MESSAGE_TYPE	A VARCHAR column whose value must be <i>NET</i> for network popup message types.

RECIPIENTS	<p>A VARCHAR column whose value is the network username, network computer name, or network messaging name to send the message to. If you want to send the same message to multiple recipients you can use comma-separated list of recipient names instead of sending multiple messages.</p> <p>If you do not specify RECIPIENTS value or insert NULL value into column, DB Mail will broadcast the message to all the names in your domain or workgroup.</p>
MESSAGE	A VARCHAR column whose value is the actual message.

Example:

```
INSERT INTO dbmail.pipe (message_type, recipient, message)
VALUES ('NET', 'MY COMPUTER NAME' 'Test message, please ignore.');
```

CHAPTER 4, Installation and Uninstallation

This section describes the installation and de-installation procedures. Once the software installation is complete, DB Mail objects should be installed on the Database side. This is explained in later sections.

Installing DB Mail is a relatively simple task, provided the system requirements are met. We will look at some of the details of performing the installation in this chapter.

Front-end Installation

DB Mail Server installation

The DB Mail Setup program provides a straightforward interface for DB Mail installation. When prompted what you want to install choose DB Mail Server option and then simply follow the Setup Wizard that will guide you through the entire installation process. When installing on a Windows NT environment, you should be logged on using an account that is a member of the local Administrators group before you install the program.

Please note the following points with regard to the installation:

- Keep the License Key supplied by Soft Tree Technologies handy. This will need to be entered during the installation.
- If you are installing DB Mail on a trial basis, leave the DB Mail License Key field blank when prompted. This will provide you a temporary license for a 30-day trial period. The License can then be purchased from Soft Tree Technologies and entered in using the 'Help -> License Key Code Maintenance' menu item later. If the Trial license has expired by then, you can also register using the 'Register' button on the pop-up screen after invoking DB Mail Server Console.
- By default, the Setup will install the DB Mail Server shortcut in the Startup folder. In case you are using NT 4.0 and above, you may always skip this step and install DB Mail as a Service. Refer to the APPENDIX B. Running DB Mail as a Windows NT service later in the manual.
- If you have a previous version of DB Mail installed on your system (version 2.0 or 2.1), the installation of DB Mail 2.5 will upgrade your previous installation. Run the Database Setup Wizard after the installation to install/upgrade DB Mail database side objects.
- If you have DB Mail version 1.0 installed on your system, installation of DB Mail 2.5 will NOT override your previous installation. It is advisable to de-install the previous version using the de-installation procedure for the previous version. You should also de-install the DB Mail Database side objects from databases where it was previously installed.
- At the end of the software installation to disk, you will be prompted to start the DB Mail database setup. You will need to be prepared with the appropriate Database administrator account and password as well as the connection details for the required Database(s). The Database setup will need to be performed on every database that you intend to send Email messages from using DB Mail. If you chose not to install the Database Setup at this time, or you need to install the Database side objects on other databases, you can invoke the 'Database Setup' program from the DB Mail 2 program group at a later time.
- If you are going to use the ODBC interface, configure desired data sources using the ODBC Administrator found in the Windows Control Panel. See the following sections for details.

**Important notes:**

During software installation of DB Mail, you will be asked to install the database side objects. If you choose not to install these objects at that time, you may do so later using DB Mail Database Setup utility. For more information on database side installation refer to the [Back-end Installation](#) topic.

VoMS installation

VoMS software installation is only required if you are planning to use DB Mail Voice Messaging functions. In that case you must install VoMS server software on any Windows 98 or better computer on your network featuring a sound board and either internal or external voice modem or an Intel Dialogic phone board. This could be the same computer that is running the DB Mail Server or any other computer on your network.

The DB Mail Setup program provides a straightforward interface for VoMS installation. When prompted what you want to install choose Voice Messaging Services option and then simply follow the Setup Wizard that will guide you through the entire installation process. When installing on a Windows NT environment, you should be logged on using an account that is a member of the local Administrators group before you install the program.

Please note the following points with regard to the installation:

- Keep the License Key supplied by Soft Tree Technologies handy. This will need to be entered during the installation.
- If you are installing VoMS on a trial basis, leave the Voice Message Server License Key field blank when prompted. This will provide you a temporary license for a 30-day trial period. The License can then be purchased from Soft Tree Technologies and entered in later.
- By default, the Setup will place VoMS server shortcut into the Startup folder.
- At the end of the software installation to disk, you will be prompted to start the VoMS server and use the VoMS Administrator utility to configure modem connection settings and other voice messaging properties. If you chose not to configure Voice Messaging Services at this time, you can invoke the 'Administrative Console' shortcut from the DB Mail 2 \ Voice Message Server program group at a later time.

**Important notes:**

By default VoMS server is installed without a password meaning that anyone on your network who has VoMS Administrator utility installed cannot connect to the VoMS server. It is highly recommended that immediately after the installation you use the VoMS Administrator utility to set a password for all administrative connections. If you choose to set a password, click the **Password** button available on the **Users** screen. Enter new password and press the OK button to save it. The password will be stored in encrypted format in the system registry and all consecutive connections to the VoMS server will require the same password entered on the VoMS Administrator connection dialog.

Back-end Installation

Requirements

DB Mail requires DB Mail database objects to be created in every database from where you want to send email, fax and other electronic messages. To install and uninstall these objects as well as manage user access to various DB Mail features you can use DB Mail Database Setup Wizard. The wizard can be started using any of the following methods:

- Run DB Mail installation Program and choose Database Setup option.
or
- From DB Mail Server Console menu select File/Install Database Objects
or
- From DOS command prompt, change current directory to DB Mail home directory and then run `DB_MAIL /SETUP` command.

This will start DB Mail Database Setup Wizard, which will guide you through the installation process.



The following sections describe what needs to be installed on different database systems.

Oracle

The main objects include the DB_MAIL package, the MAILSTORE directory object (for Oracle 8i and above) as well as a table and a sequence for supporting attachments to email messages and electronic faxes. These objects need to be installed in all the databases, regardless of version, that users want to send mail from.

DB_MAIL is implemented as the one single PL/SQL package that encapsulates all of the messaging calls. This package is installed in the SYSTEM schema and a public synonym with the same name is created so that users can call DB_MAIL package functions without directly referring to the SYSTEM schema.

To install DB Mail database objects in an Oracle database perform the following steps:

1. Start DB Mail Database Setup utility.
2. Choose **Oracle 7.3, 8.x, 9.x, and later** option and then click the **Next** button.
3. Enter valid password for the SYS user into the **Password** field.
4. Select valid Oracle TNS name from the **Host String** drop-down list. If you cannot find an existing name in the list simply type in a valid TNS entry name then click the **Next** button. When installing

DB Mail on a local Oracle instance you can leave the **Host String** field blank.

5. (Oracle8i and later only) Type in name of an existing directory where DB Mail can create temporary files for email attachments and electronic faxes. If your Oracle database is already configured for external file access you can enter name of one of the directories listed in UTL_FILE_DIR initialization parameter. If you do not have UTL_FILE_DIR parameter configured yet you will need to add that parameter to your Oracle parameters file.



Important notes:

After you modify your Oracle parameter file, you will need to stop and then restart your database instance in order for the changes to take effect. Failure to do so will prevent DB Mail from supporting email attachments and electronic faxes.

6. Click the **Finish** button. DB Mail Database Setup will run installation scripts compatible with your Oracle database version.

Access Privileges

Because DB_MAIL package may call functions in DBMS_PIPE, UTL_FILE and DBMS_BACKUP_RESTORE (Oracle 9i and later only) packages, the SYSTEM user needs EXECUTE privilege on these system packages.

A user (other than SYSTEM) who needs to send messages from within the database also needs access to the DB_MAIL package. Unless you want to restrict the privilege of sending messages from within the database to specific individuals, you should grant PUBLIC access to EXECUTE the DB_MAIL package and SELECT/INSERT/UPDATE/DELETE privilege to PUBLIC on the MAIL_ATTACH table.

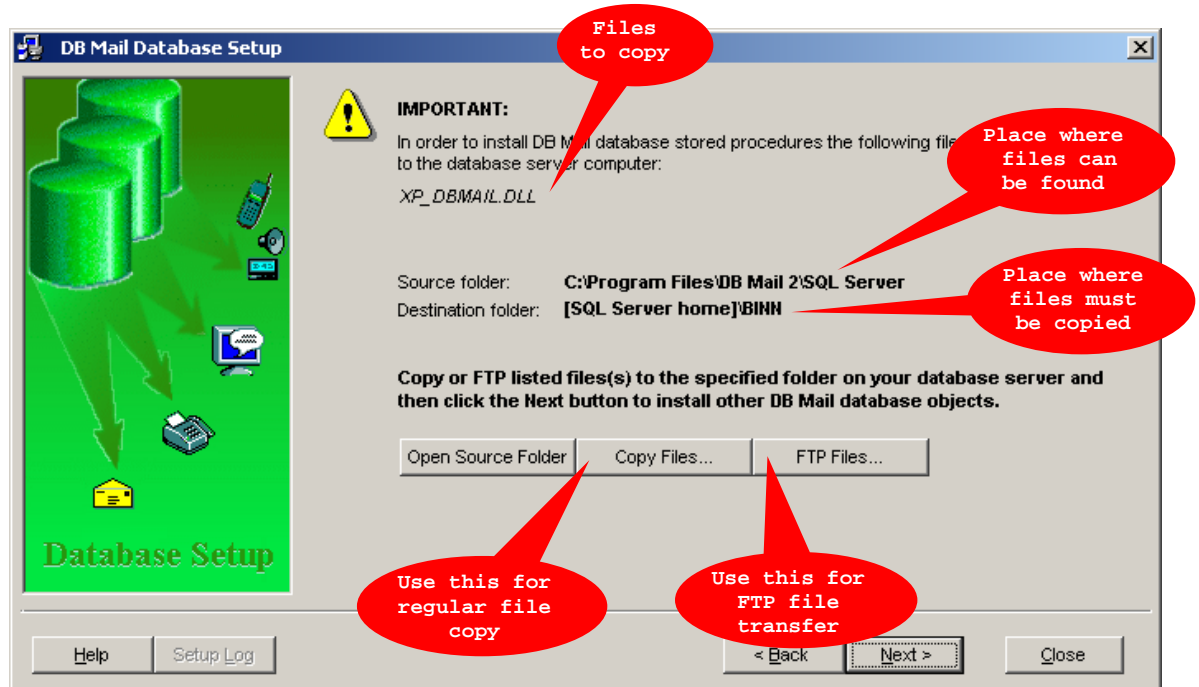
Microsoft SQL Server

DB Mail server side objects are installed in the **master** database within the DBMAIL schema. In order to create this schema, the DB Mail Database Setup utility creates the DBMAIL login and database user. In that schema the setup creates three database tables: PIPE, ATTACH, ATTACH_SEQ tables and also a set of stored procedures for sending emails, faxes, page and text messages, administrative alerts, and network popup messages. The setup will also install the *xp_dbmail.dll* on your SQL server computer that is required for DB Mail extended stored procedures.

These objects need to be installed in all Microsoft SQL Servers, regardless of version, that users want to send mail from.

To install DB Mail database objects in Microsoft SQL Server, perform the following steps:

1. Start DB Mail Database Setup utility.
2. Choose **Microsoft SQL Server 6.5, 7, 2000, and later** option and then click the **Next** button.
3. Enter valid password for the SA user into the **Password** field.
4. Select valid server name from the **Server** drop-down list. If you cannot find an existing name in the list simply type in the name then click the **Next** button. When installing DB Mail on a local SQL Server instance you can leave the **Server** field blank or type in *(local)* as a server name.
5. Use **Copy Files** or **FTP Files** option to copy *xp_dbmail.dll* file to BINN directory on your SQL Server computer.



For instructions on how to copy files see [How to copy files \(SQL Server example\)](#) topic.

For instructions on how to FTP files see [How to FTP files \(DB2 example\)](#) topic.

6. After you are done with copying files click the **Next** button again and then click the **Finish** button. DB Mail Database Setup will run installation scripts compatible with your SQL Server version.

Sybase SQL Server, ASE, ASA

DB Mail server side objects are installed in the **sybssystemprocs** database within the DBMAIL schema. In order to create this schema the DB Mail Database Setup utility creates the DBMAIL login and database user. In that schema the setup creates three database tables: PIPE, ATTACH, ATTACH_SEQ tables and also a set of stored procedures for sending emails, faxes, page and text messages, administrative alerts, and network popup messages. The setup will also install the *xp_dbmail.dll* on your SQL server computer that is required for DB Mail extended stored procedures.

These objects need to be installed in all Sybase SQL Servers, regardless of version, that users want to send mail from.

To install DB Mail database objects in Sybase SQL Server, perform the following steps:

1. Start DB Mail Database Setup utility.
2. Choose **Sybase SQL Server, Sybase ASE 10.x, 11.x, 12.x, and later** option and then click the **Next** button.
3. Enter valid password for the SA user into the **Password** field.
4. Select valid server name from the **Server** drop-down list. If you cannot find an existing name in the list simply type in the name then click the **Next** button. When installing DB Mail on a local SQL Server instance you can leave the **Server** field blank or type in *(local)* as a server name.
5. Select which version of DB Mail you want to install. If your Sybase server does not support Java you should install the limited version of DB Mail that does not support email attachments and fax procedures.

If your Sybase server supports Java but Java is not currently enabled you need to execute **sp_configure "enable java", 1** command using ISQL or any SQL editor. Then shut down and restart the server. You can also use Sybase Central console to change server configuration parameters.

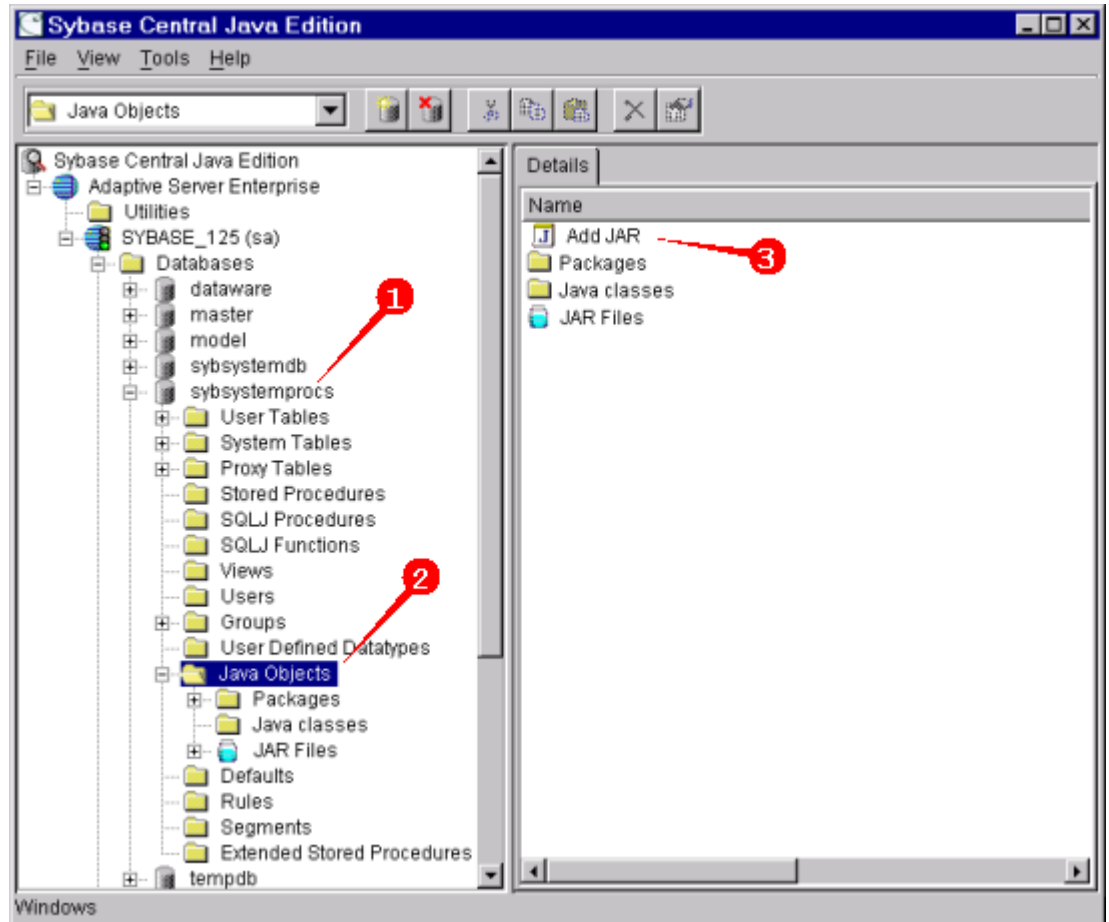


Important Notes:

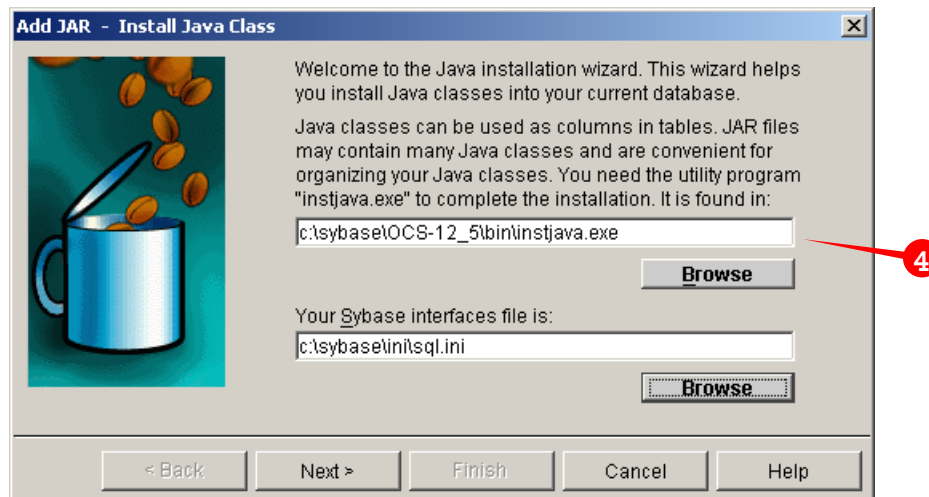
- By default, Sybase Adaptive Server Enterprise is not enabled for Java. You cannot install Java classes or perform any Java operations until the server is enabled for Java.
- You can increase or decrease the amount of memory available for Java in Adaptive Server and optimize performance using *sp_configure* system stored procedure. Java configuration parameters are described in the Sybase System Administration Guide.

If DB Mail Database fails to load DBMAIL.JAR file into your Sybase data you can load this file manually using Sybase Central Java Edition as shown below:

1. In a Sybase Central Java Edition installation, connect to the database server and then select and expand **sybssystemprocs** database.
2. Select **Java Objects** folder.
3. Double-click **Add JAR** icon.

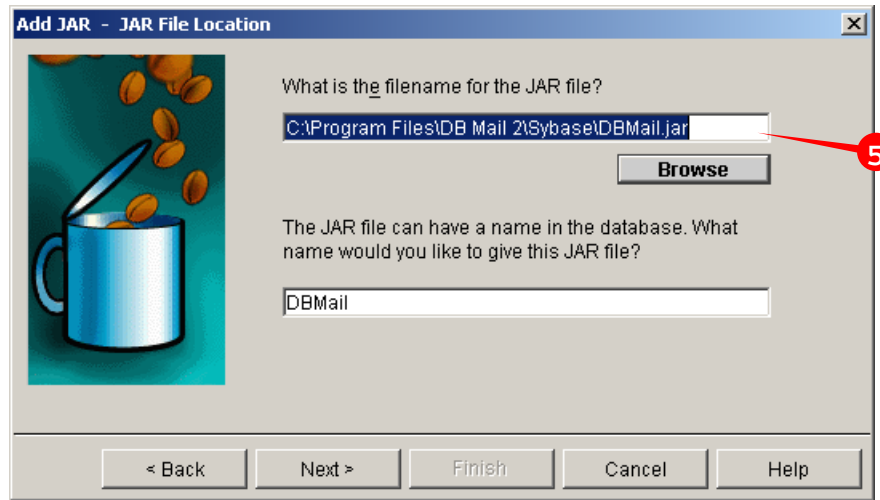


4. The Add JAR – Install Java class dialog will appear. If default settings in the dialog look ok click the **Next** button, otherwise correct setting and then click the **Next** button.

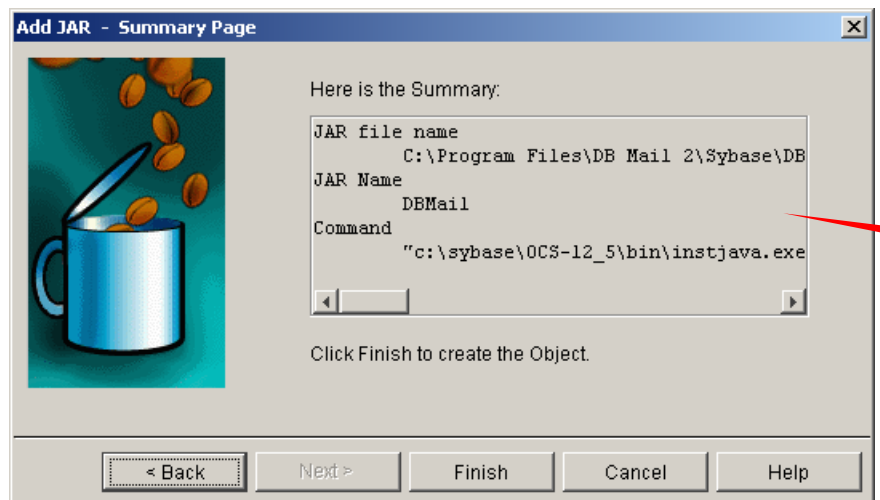


5. Use the browse button to locate the *DBMail.jar* file. By default this file can be found in *C:\Program Files\DB Mail 2\Sybase* directory. Click the **Next**

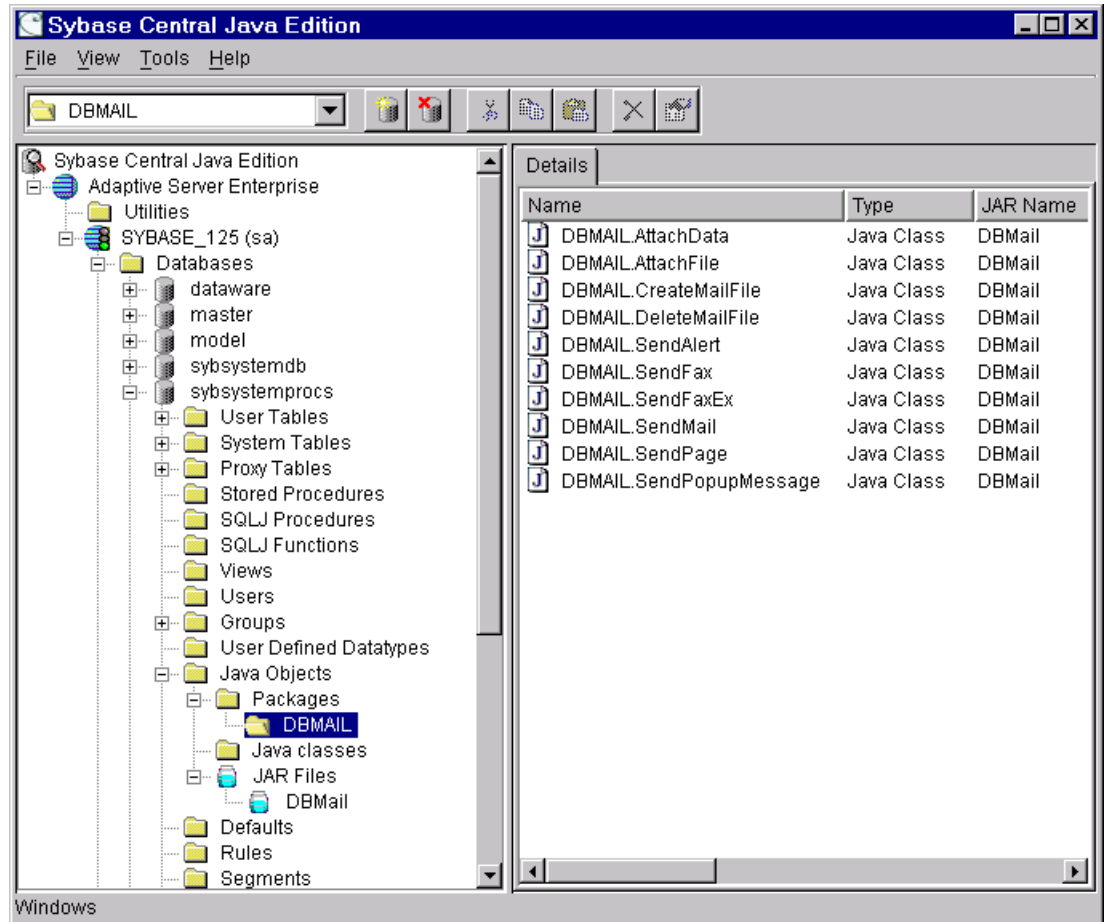
button again.



6. Verify selected options and click the **Finish** button.



7. If the JAR file loads correctly you should be able to see DB Mail Java classes loaded into the database as on the following screen shot.



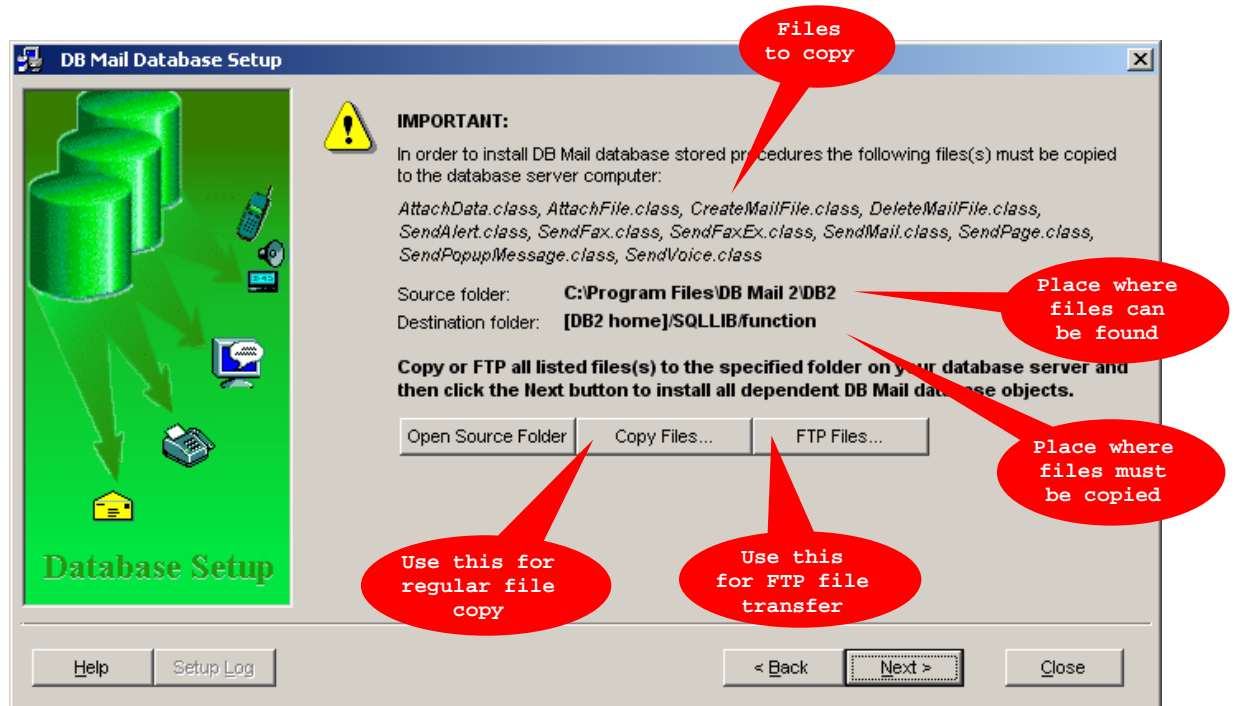
IBM DB2

DB Mail server side objects are installed in every DB2 database from where users will be sending mail. In order to install back-end objects the DB Mail Database Setup utility creates the DBMAIL schema. In that schema the setup creates three database tables: PIPE, ATTACH, ATTACH_SEQ tables and also a set of Java stored procedures for sending emails, faxes, page and text messages, administrative alerts, and network popup messages.

These objects need to be installed in all DB2 databases, regardless of version, that users want to send mail from.

To install DB Mail database objects in DB2 database perform the following steps:

1. Start DB Mail Database Setup utility.
2. Choose **IBM DB2 6.1, 6.2, 7.x, 8.x and later** option and then click the **Next** button.
3. Enter valid user id and password for a database administrator account into the **User** and the **Password** fields.
4. Select valid ODBC Profile name from the **Server** drop-down list. If you cannot find an existing name in the list simply type in the name then click the **Next** button.
5. Use **Copy Files** or **FTP Files** option to copy DB Mail Java classes to *DB2/SQLLIB/function/DBMAIL* subdirectory on your DB2 computer.



For instructions on how to copy files see [How to copy files \(SQL Server example\)](#) topic.

For instructions on how to FTP files see [How to FTP files \(DB2 example\)](#) topic.

- After you are done with copying files click the **Next** button again and then click the **Finish** button. DB Mail Database Setup will run installation scripts compatible with your DB2 database version.

How to copy files (SQL Server example)

If you can access SQL Server home directory using a network share click the **Copy Files** button. The standard Windows **Browse for Folder** dialog will appear.

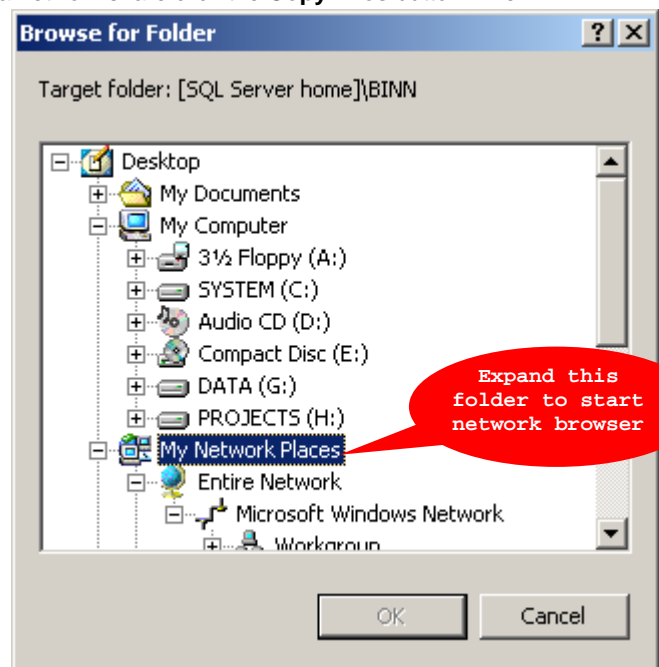
If you can access SQL Server computer via network browsing, expand the special "My Network Places" folder (On Windows 95/98/NT 4 this folder is called "Network Neighborhood"). Within that folder, open "Entire Network" and "Microsoft Windows Network" as needed. You should see a list of workgroups and network domains; open the domain or workgroup your SQL Server belongs to. Locate your SQL Server computer and then within that computer locate SQL Server installation directory. Inside that directory double-click on the BINN folder. DB Mail will copy required files over the network to the BINN directory.

If you cannot access SQL Server computer via network browsing you still should be able to access it via a mapped network drive. To map a remote SQL Server drive as a network drive use the "Map Network Drive" command, which you can access by right-clicking on the "My Computer" icon displayed on the Desktop. Type in the Windows-style share name of your directory, for example:

```
\\myserver\system
```

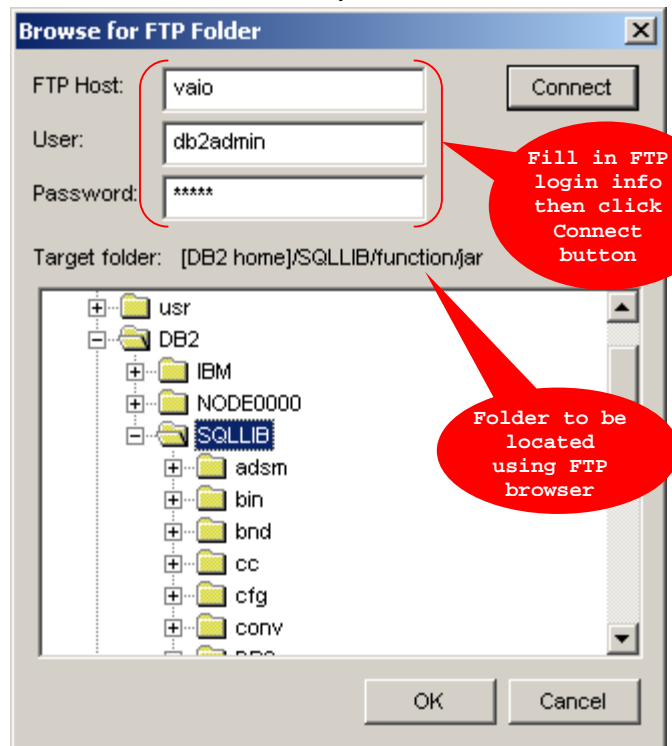
At this point, you must authenticate yourself before you are granted access. Enter appropriate user name and password for accessing the remote computer and then click the Enter key. Please note that you may need to use user name and password that are different from your database user id and password. Please contact your network administrator if you do not have your password or need additional help with this procedure.

After you mapped remote drive you can copy files to it just as you would copy files to your local drive.



How to FTP files (DB2 example)

If you cannot use a mapped drive but have FTP server running on your SQL Server computer you can use the **FTP Files** option. The setup utility features built-in mini FTP browser and FTP file copy utility. To start the FTP browser click **FTP Files** button and then enter FTP server connection parameters and press **Connect** button. Please note that you may need to use user name and password that are different from your database user id and password. Please contact your network administrator if you do not have your password or need additional help with this procedure.



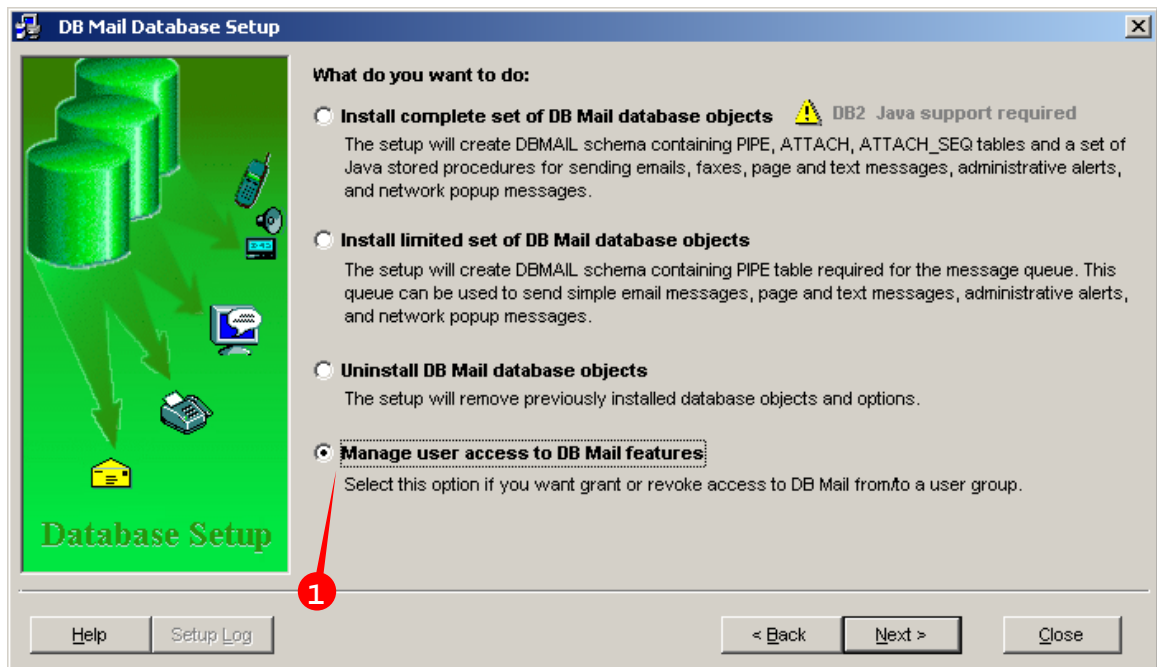
After the setup connects to your FTP server locate and expand *DB2* home directory. Within that directory locate *SQLLIB/function* subdirectory. Click the **OK** button. DB Mail will create the *DBMAIL* subdirectory and then FTP all required files to the new *DBMAIL* subdirectory.

Managing user access to DB Mail features

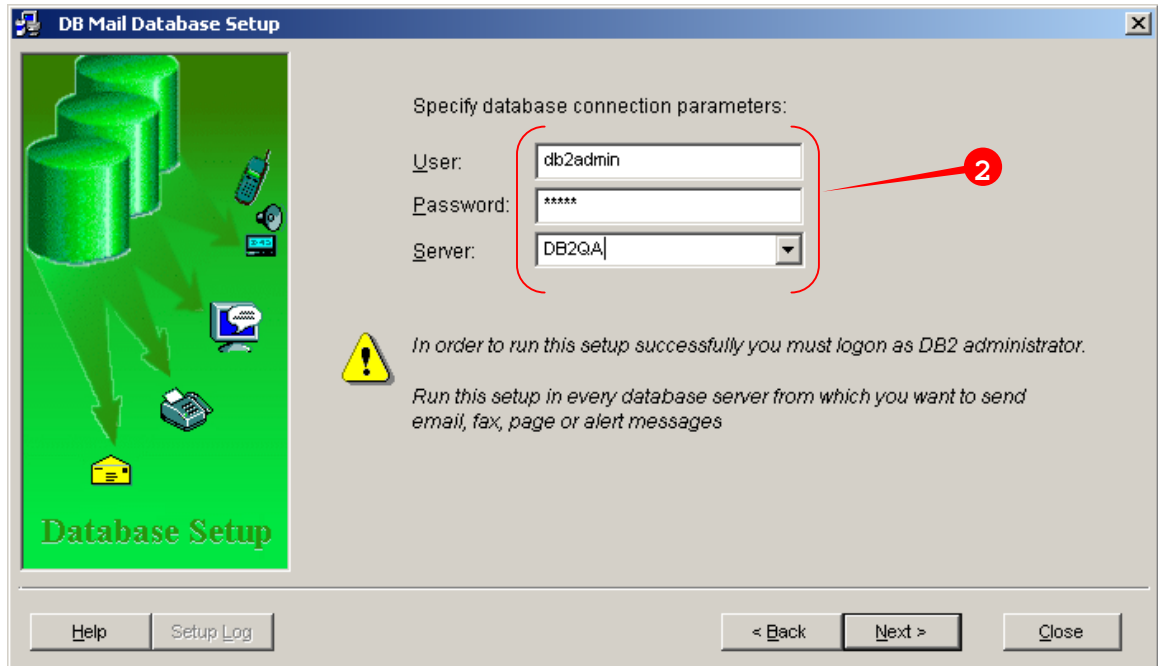
Finally, a word about accesses privileges. The DB Mail objects are installed using database administrator's privileges. A user (other than a database administrator) who needs to send messages from within the database needs access to the DB Mail objects. Unless you want to restrict the privilege of sending messages from within the Database to specific individuals, you should grant PUBLIC access to DB Mail objects. To manage user access privileges use DB Mail Database Setup utility described in the By default VoMS server is installed without a password meaning that anyone on your network who has VoMS Administrator utility installed cannot connect to the VoMS server. It is highly recommended that immediately after the installation you use the VoMS Administrator utility to set a password for all administrative connections. If you choose to set a password, click the **Password** button available on the **Users** screen. Enter new password and press the OK button to save it. The password will be stored in encrypted format in the system registry and all consecutive connections to the VoMS server will require the same password entered on the VoMS Administrator connection dialog.

Back-end Installation topic.

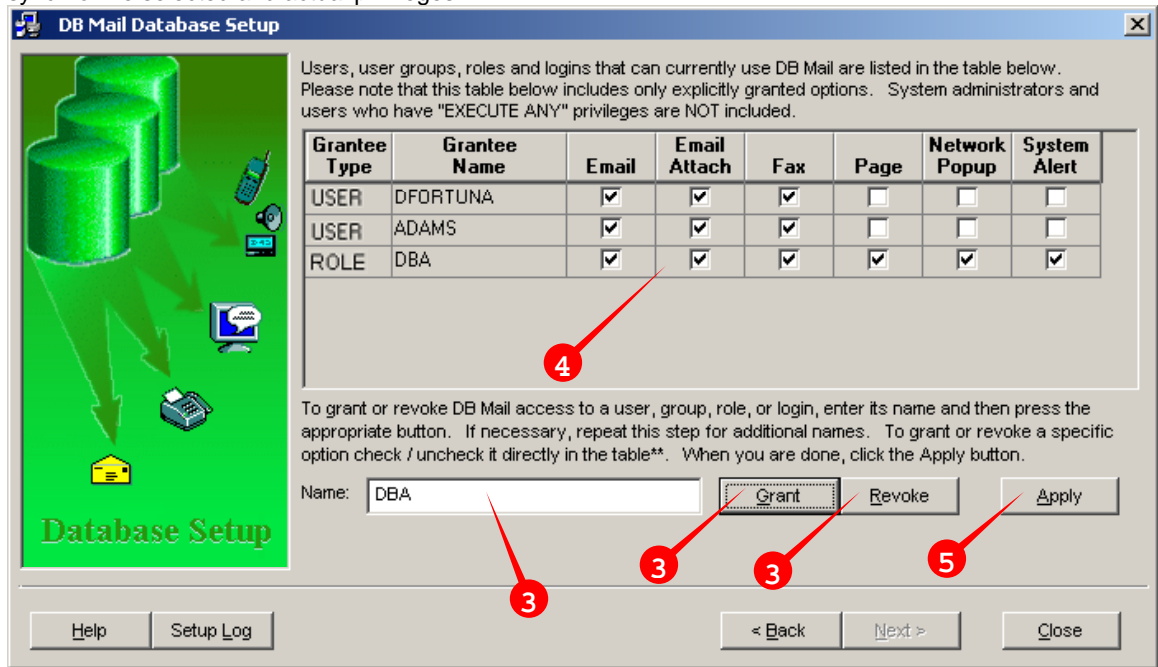
1. When prompted, choose **Manage user access to DB Mail features** option and then click the **Next** button.




2. Complete database connection parameters and then click the Next button again.



3. Enter database user names into the **Name** field one by one each time pressing the **Grant** button. If you made a mistake and would like to remove a user name that was entered by mistake, type that name in the **Name** field and then press the **Revoke** button.
4. In the Access Privileges table check/uncheck access options as appropriate.
5. Press the Apply to apply selected privileges. DB Mail Database setup will automatically synchronize selected and actual privileges.



 **Tip:** You can change user access privileges at any time by rerunning DB Mail Database Setup utility and choosing "Manage User Access" option again. The Database Setup utility will automatically

retrieve and show existing access privileges.

Testing

After you complete database-side installation the DB Mail Setup Wizard will offer to test the installed DB Mail messaging interface. If you choose to test it, the Test DB Mail Database Setup dialog will appear. Using this dialog you can test messages of different types. You can test all messages at once or you can pick any combination of messages.

To run tests:

1. Check which messages you want to send.
2. Enter message recipient numbers and names.
3. Click the **Test** button

DB Mail will create selected messages by running appropriate SQL statement in the database.

If you get any errors make sure to read additional test information shown on the **Test Results** tab page, if necessary, use scrollbars to scroll the displayed test results.



Important notes:

DB Mail Server Console must be used to configure database connection profiles and messaging options if you want to check how test messages get delivered to their destinations. If you do not do so before or during the testing phase your test messages will be saved in the message queue and will remain there until you run DB Mail Server Console and configure database connection profiles and messaging options. For more information on setting DB Mail configuration options see [CHAPTER 5, Configuring DB Mail](#).

Uninstallation


The DB Mail supports standard uninstallation mechanism for removing program files from the system where DB Mail was previously installed.

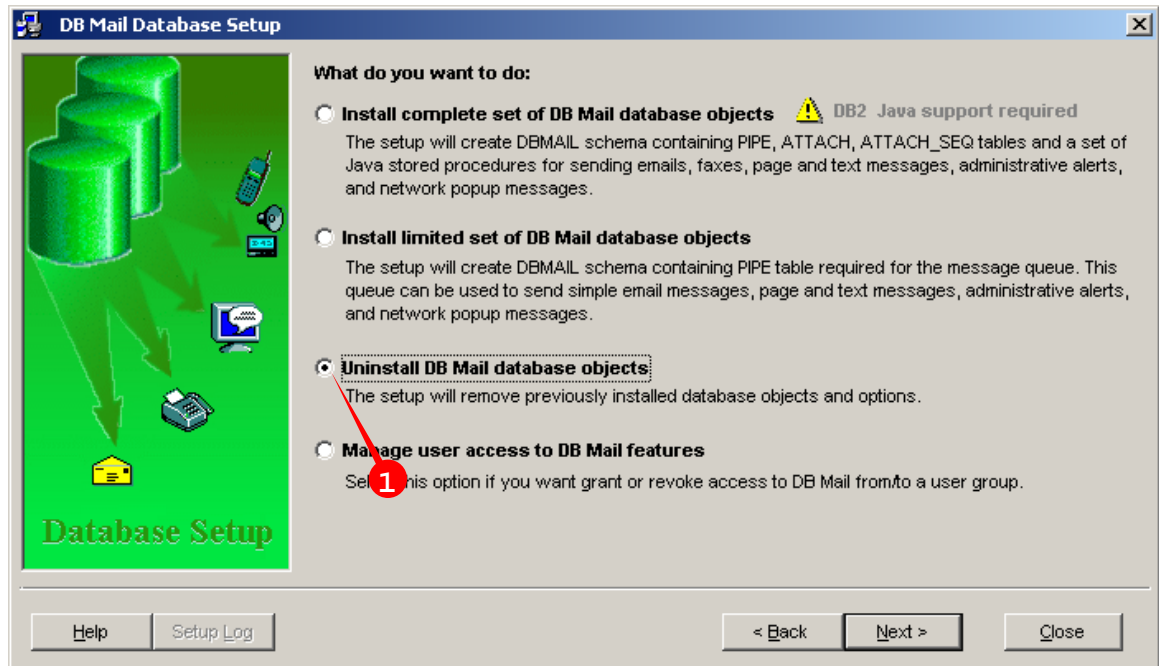
To uninstall DB Mail do the following:

1. Click Windows **Start** button, from the Start Menu select **Settings**, then **Control Panel**.

2. Double-click **Add/Remove Programs**.
3. Select the **DB Mail 2** item in the programs list, click **Add/Remove** button

Delete all files that are left behind in the DB Mail home directory. Delete the home directory.

 **Important Note:** If you wish to uninstall DB Mail database back-end objects you should run DB Mail Database Setup utility before uninstalling DB Mail server components.



After you uninstall main DB Mail objects using DB Mail Database Setup utility you may also want to delete additional files copied during the Installation procedure.

Microsoft SQL Server

If you uninstall DB Mail from Microsoft SQL Server, do not forget to manually delete xp_dbmail.dll file from SQL Server BINN directory.

DB2

If you uninstall DB Mail from IBM DB2 server, do not forget to manually delete all *.class files from `[DB2 home]/SQLLIB/function/DBMAIL` subdirectory on your DB2 server.

CHAPTER 5, Configuring DB Mail

Before you can use DB Mail you must configure database connections and email properties. To change DB Mail configuration, select **File/Options** item in the DB Mail Server Console menu. The DB Mail Options dialog will appear. This dialog contains 10 tab pages for different DB Mail options. They are:

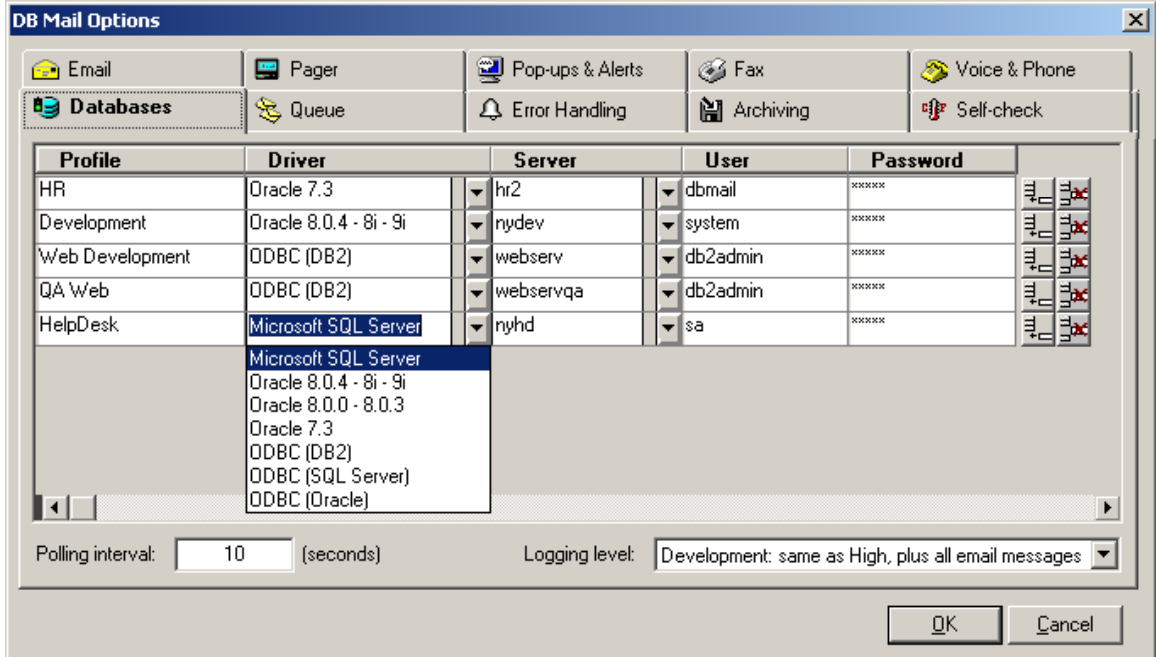
- Database
- Queue
- Error Handling
- Self-checking
- Archiving
- Email
- SMS & Pager
- Pop-up & Alerts
- Fax
- Voice & Phone


You will need to configure these options carefully for DB Mail to work properly and effectively. To better explain the features, screen shots are presented along with explanations for the options.




Configuring Databases Options

Database connections for DB Mail are specified here. Before you add a new database connection, make sure you have already completed installing DB Mail in your database(s). See the DB Mail By default VoMS server is installed without a password meaning that anyone on your network who has VoMS Administrator utility installed cannot connect to the VoMS server. It is highly recommended that immediately after the installation you use the VoMS Administrator utility to set a password for all administrative connections. If you choose to set a password, click the **Password** button available on the **Users** screen. Enter new password and press the OK button to save it. The password will be stored in encrypted format in the system registry and all consecutive connections to the VoMS server will require the same password entered on the VoMS Administrator connection dialog.

Back-end Installation topic for details. The details of all the databases where you would like users to send email using DB Mail will need to be captured here, and Profiles assigned to them.



Configuration Option	Description
Profile	Database profile is the name (up to 50 characters long) for the set of parameters that define a connection to a particular database. Enter any descriptive name. When logging various errors and messages DB Mail uses this profile name as a reference name for the message source.
Driver	<p>Specifies which database driver/connection method to use when connecting to your database. You should select the driver that matches your DBMS version.. Please refer to CHAPTER 2. Connecting To Your Database for more details of the various driver choices and what should be selected. The drop down box in the graphic above shows all the supported drivers.</p> <p> Important note:</p> <p>When choosing a native database driver for the connection, make sure that you select the driver that matches the version of your database client software installed on the system, not the version of your database server software. For example, if you have Oracle SQL*Net 2 with Oracle 7.3 client files installed on your workstation but connect to an Oracle 8i database, you should select OR7 Oracle 7.3 driver for the connection.</p>
Server	The server name or server connection string specifying parameters that the database driver uses to connect to the database server. For ODBC data sources specify data source name. For native database drivers it should be the same string that you use when connecting from SQL*Plus, ISQL and other database utilities. For more details on setting up database connection parameters, see CHAPTER 2. Connecting To Your Database .
User	A valid login ID for your database server that has full access to the

	objects required by DB Mail.
Password	The login password of your database server. The password you type is displayed in asterisks (*).
 button	Click this button to add a new row to the list of database connections
 button	Click this button to remove the selected row from the list of database connections
Polling interval	<p>Use this property to specify how often you want the daemon process to perform the check for new email messages placed to the DB_MAIL pipe.</p> <p>A note of caution: The DB Mail daemon process can check for new messages virtually every second. However, frequent checking will cause additional network and database load. If you are not planning to send large amounts of email messages every day, set this to a more reasonable value, such as 60 seconds.</p>
Logging level	<p>Logging level specifies the amount of information you want DB Mail to log for auditing and troubleshooting purposes. The following 3 levels are available:</p> <ul style="list-style-type: none"> • Normal – This level includes various status messages and all email processing errors. • High – This level includes all messages that are provided with the Normal level as well as some additional messages describing DB Mail progress of work. You should use this level when troubleshooting DB Mail. • Development – This level includes all messages that are provided with the High level as well as complete text of all email, fax, page and other messages sent via DB Mail. You should use this level when troubleshooting your SQL code that generates these messages. <p> Important notes:</p> <p>DB Mail logs all selected messages to the screen and DB_MAIL.LOG file. The screen buffer is limited to 1000 lines produced by the most recent messages. The DB_MAIL.LOG file is located in the same directory where the DB Mail software has been installed (\Program Files\DB Mail2 by default) and has no size limit. It should be deleted or archived from time-to-time; otherwise it may easily grow large leading to disk space usage problems. Note that higher logging levels allow more information written to the log file thus causing the log file to grow faster. You should use High and Development levels only when you are troubleshooting message processing problems and use Normal or Low levels at all other times.</p>

 **Important notes:**

Specifying the database administrator's id and password in the Profile may compromise Security, as the passwords are stored in the registry of the DB Mail server. To avoid this, you may want to create a specific user on the remote database who has limited access only to the DB Mail objects named previously.

Configuring Email Options

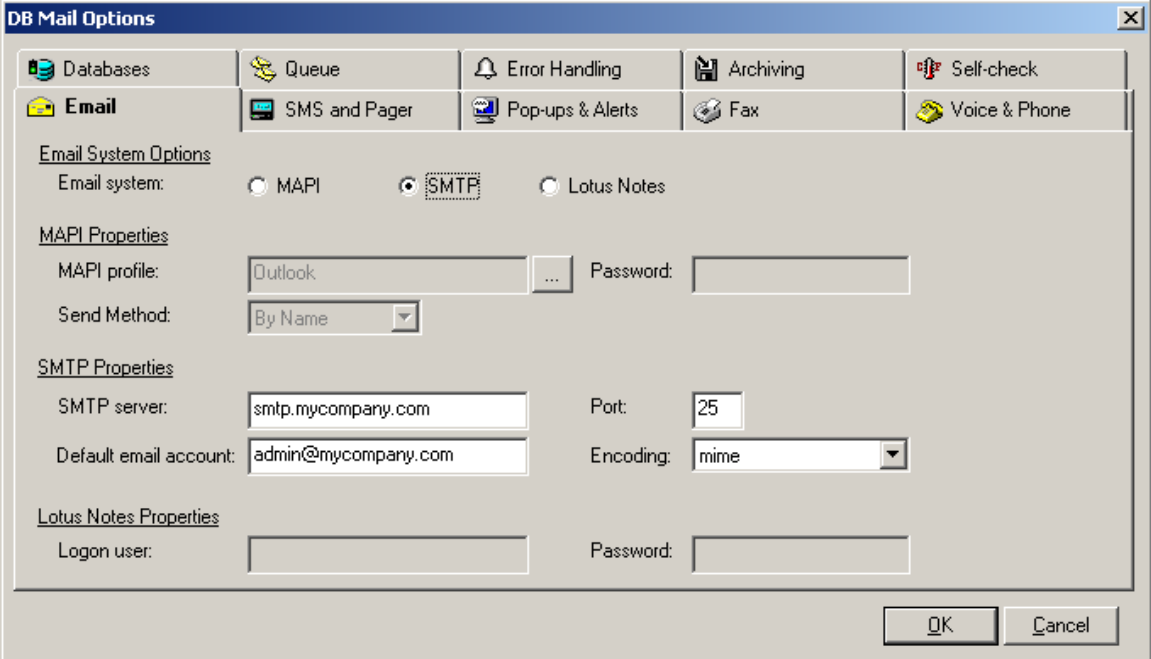
DB Mail supports three different email interfaces: standard Windows MAPI (Messaging Application Programming Interface), SMTP (Simple Mail Transfer Protocol), and Lotus Notes interface utilizing Lotus Notes API. DB Mail uses the selected interface when sending email messages.

Important Note:

Only one of these three interfaces can be selected at any time. The DB Mail Server will need to be restarted if this option is changed.

Tip:


If your telephone company supports an email-to-pager service, you can also configure DB Mail to page messages to alphanumeric pagers by sending them to the appropriate email address. Do not confuse this option with the DB Mail paging service, which uses SNPP protocol over TCP/IP network to send numeric and text messages to pagers and cell phones.



The screenshot shows the 'DB Mail Options' dialog box with the 'Email' tab selected. The 'Email System Options' section has three radio buttons: 'MAPI', 'SMTP' (which is selected), and 'Lotus Notes'. Below this, there are sections for 'MAPI Properties', 'SMTP Properties', and 'Lotus Notes Properties'. In the 'MAPI Properties' section, the 'MAPI profile' is set to 'Outlook' and the 'Send Method' is 'By Name'. In the 'SMTP Properties' section, the 'SMTP server' is 'smtp.mycompany.com', the 'Port' is '25', and the 'Default email account' is 'admin@mycompany.com'. The 'Encoding' is set to 'mime'. In the 'Lotus Notes Properties' section, the 'Logon user' and 'Password' fields are empty. The dialog box has 'OK' and 'Cancel' buttons at the bottom right.

The options are explained below.

Configuration Option	Description
Email system	<p>Select one of the following:</p> <ul style="list-style-type: none"> • MAPI • SMTP • Lotus Notes <p>Based on your email system selection you should complete the corresponding email system properties box. Depending on the Email system chosen, the other property boxes will be grayed out.</p>
MAPI Profile	<p>This is the name of a MAPI profile. MAPI profile contains a set of parameters that define a connection and email send method to a</p>

	particular mail server. Type the desired name or click the  button to select the name of an existing MAPI profile. If you don't have a profile yet or would like to create a new profile, use the Control Panel/Mail applet to create MAPI profile. For details on creating new MAPI profiles see the Configuring New MAPI Profile topic later in this chapter.
MAPI Password	Enter a valid password required to logon to your email client. Leave this field blank if your email client does not require a password.
MAPI Send Method	Different email clients use various internal MAPI properties when sending email messages. For example, Microsoft Outlook® expects the email address or email user name to appear in the Name property, while others such as Netscape Messenger® expect it to appear in the Address property. Select the Send Method that is compatible with your email client. Important notes: If the selected method is not compatible with your email client, an Email Compose window will appear on the screen for every email message sent using MAPI interface. To avoid this situation, select another method.
SMTP Server	Name of your SMTP server
SMTP Port	Port used by your SMTP Server for SMTP listening. The default port used by most servers is 25.
Default Email Account	Default sender email address to be used for outgoing email when sender is not specified. Important notes: Some email servers require valid email address for authentication.
Encoding	Select the desired encoding method that is supported by the recipient's email programs. Today most email programs support mime . Important notes: For email messages with attachments DB Mail ignores the selected encoding method and always uses mime . If you are going to send email messages to a pager, you should select none for the encoding.
Lotus Notes User	Valid user name that you use to logon to your Lotus Notes client software.
Lotus Notes Password	Valid password that you use to logon to your Lotus Notes client software.

 **Important notes:**

- MAPI and SMTP interfaces enable DB Mail to interact with multiple messaging systems across a variety of software and hardware platforms whereas Lotus Notes interface works with the following configurations only:
 - 1) Lotus Notes workstation v4.5 and later running on Windows NT workstation or server (Intel platforms only). There is no limitation with respect to the Lotus Notes server version and platform.
 - 2) Lotus Notes server v4.5 and later running on Windows NT server (Intel platform only).
- If you have installed Lotus Notes MAPI extensions, you can still use MAPI interface to send

and receive email via Lotus Notes. It is recommended that the MAPI interface be used wherever possible. The Lotus Notes interface may not keep up with the newer versions of the Lotus Notes although SoftTree Technologies will make all efforts to release updated versions where required.

- DB Mail Lotus Notes interface consists of two parts:
 - Notes email interface library
 - Notes extension manager

DB Mail email interface library for Lotus Notes always uses the default Notes mail database and mail server. Default settings are taken from Notes environment variables.

 **Warning:**

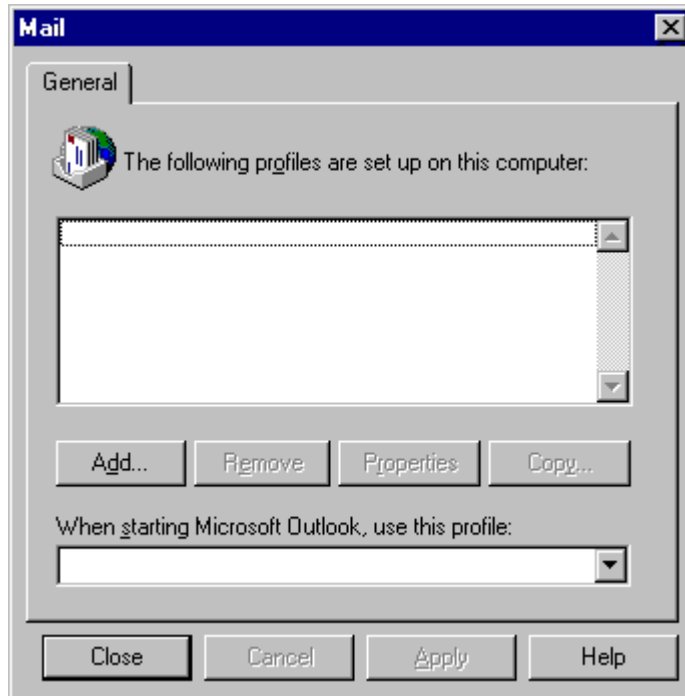
DB Mail installs Notes extension manager program that intercepts the Notes password prompt, and supplies the password that you specified in the DB Mail options. The extension manager allows sending email messages via Notes without user intervention when Notes normally requires a password. The extension manager program is built as a set of dynamic link libraries (DLLs). These DLLs are loaded by Lotus Notes on startup, and they behave as if they are part of the Lotus Notes software. While a DB Mail email operation is in progress, your Lotus Notes is exposed to other users and programs because no password is required at that moment to interact with the Lotus Notes software. Before selecting Lotus Notes interface, make sure you don't have another Notes extension manager already installed on your system. To verify this, make sure you don't have the EXTMGR_ADDINS key in the NOTES.INI file or that key is not initialized.

Configuring New MAPI Profile

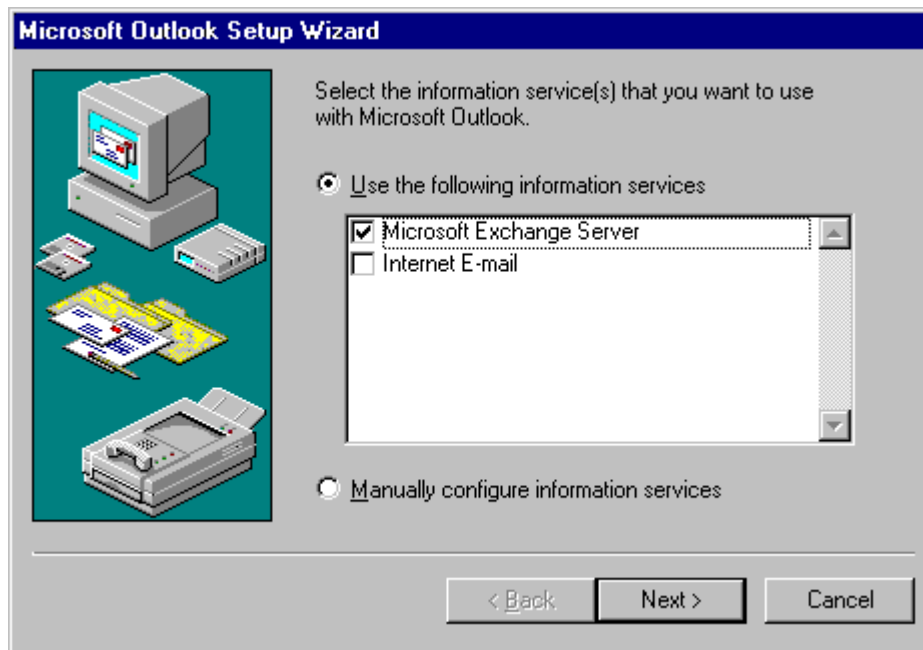
Before you configure a new or existing MAPI profile, make sure you have some email client software installed on the DB Mail computer. The following instructions refer to the configuration of MAPI for Microsoft Outlook ®, but they can be easily adopted for other email client software.

To configure MAPI with the Outlook client on the DB Mail computer follow these steps:

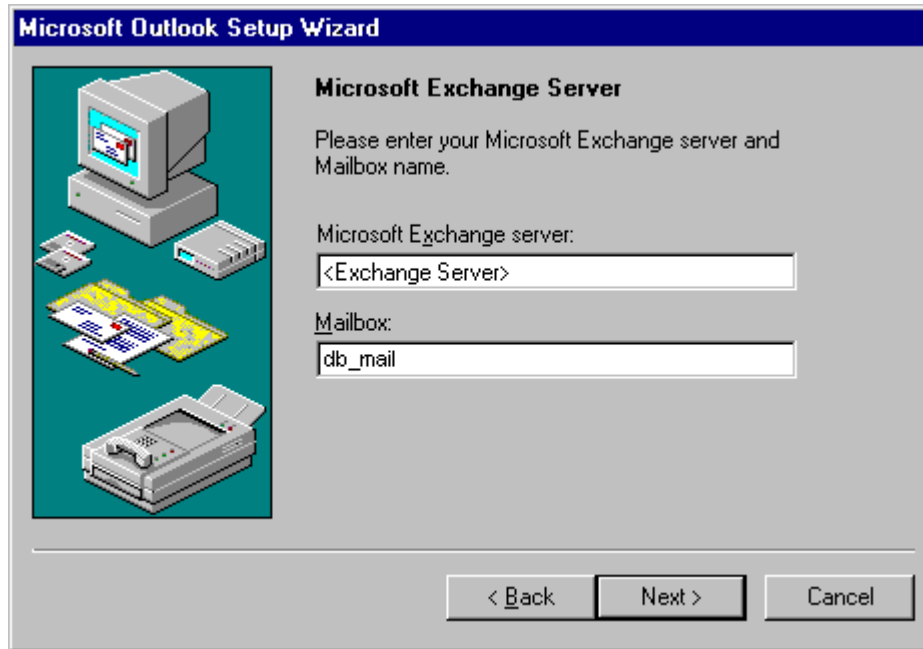
1. In Control Panel, double-click the **Mail** icon.
2. Click **Add**.



3. Select the **Microsoft Exchange Server** check box or select any other available mail server check box that appears in the **Information Services** list. Click **Next**.



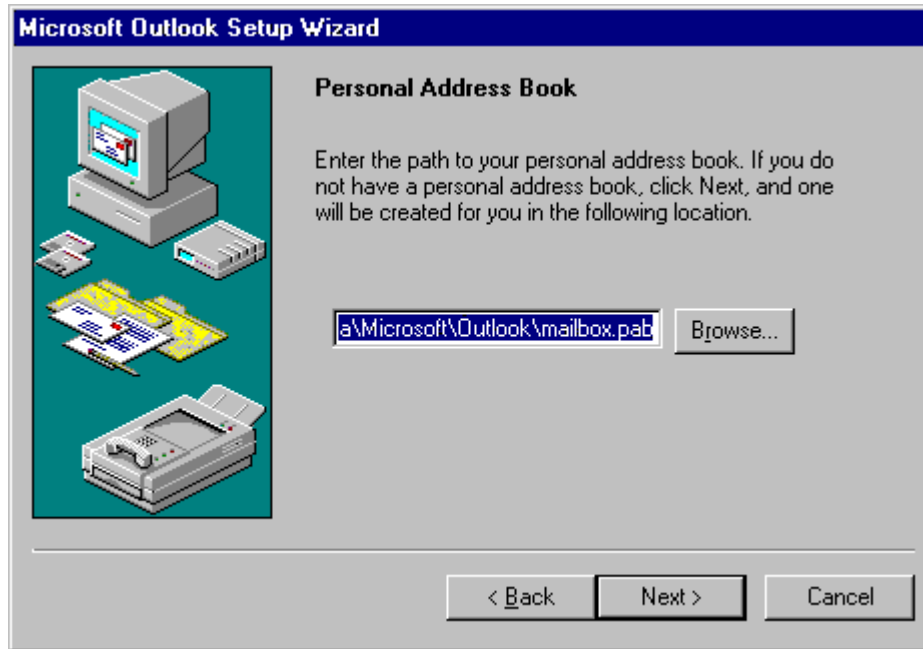
4. Type the Exchange Server name in the **Microsoft Exchange server** box, type *db_mail* or any other name in the **Mailbox** box, and then click **Next**.



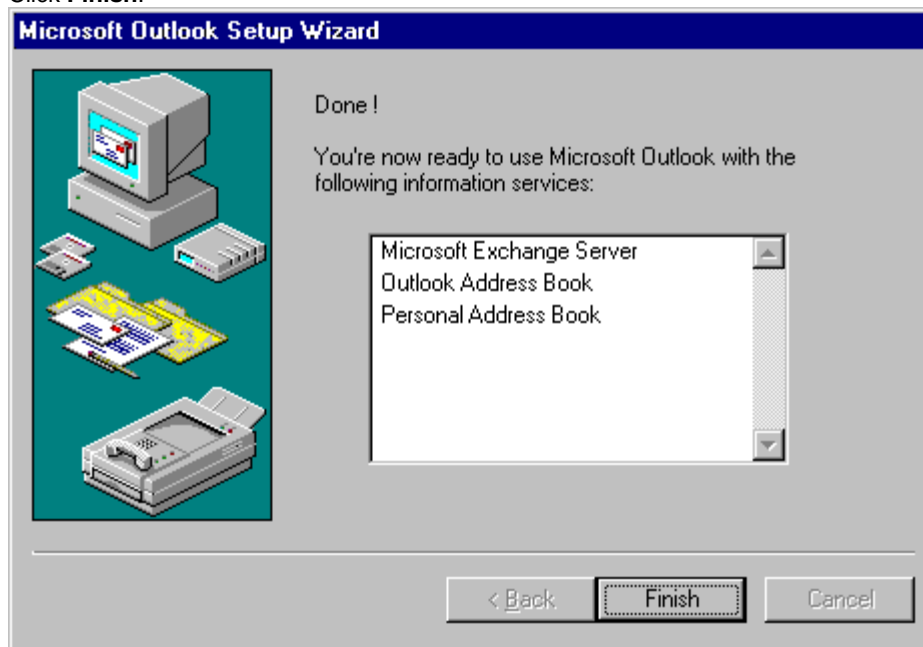
5. Verify that **No** is selected. If it is not already selected, click **No**, and then click **Next**.



6. Type the path and name of your **Personal Address Book** or choose the default provided, and then click **Next**. Microsoft recommends that you store this information in the Windows NT Profiles directory; for example, C:\Winnt\Profiles\db_mail. This will help to prevent the file from being accidentally overwritten.

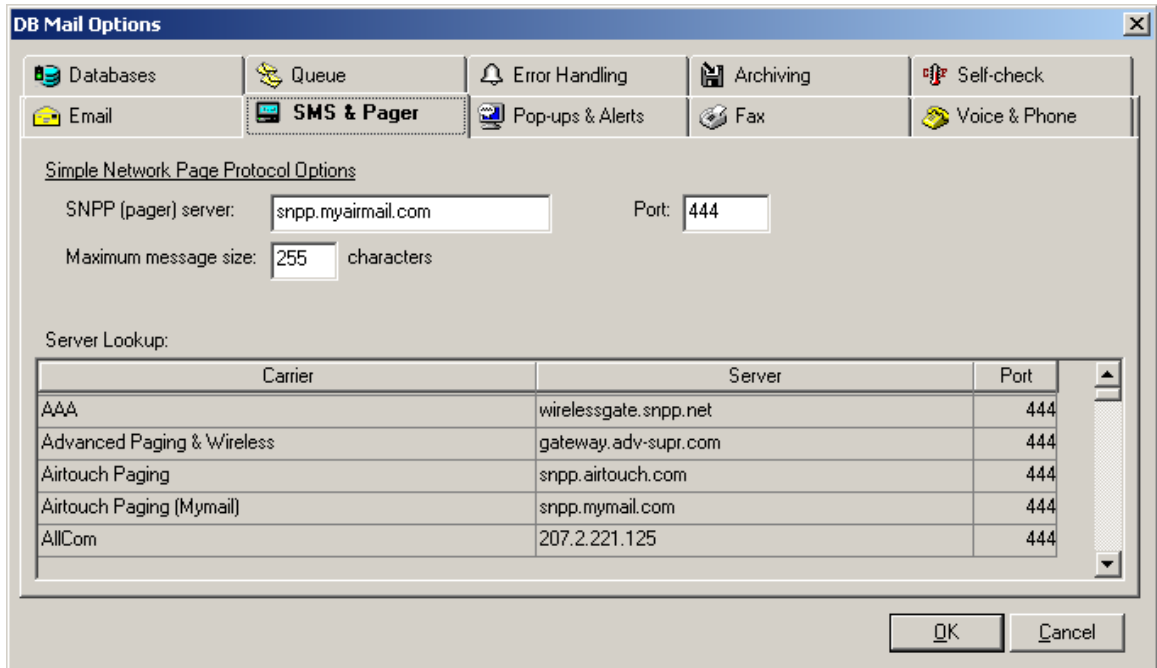


7. Click **Finish**.



Configuring SMS and Pager Options

DB Mail can be configured to directly send SMS messages to alphanumeric pagers and cell phones. This is performed using the Simple Network Page Protocol. In Oracle, the SMS interface is accessed via the SEND_PAGE function call in the DB_MAIL package. In other databases the Pager is accessed via the SendPage stored procedure or by directly inserting data into DBMAIL.PIPE table.

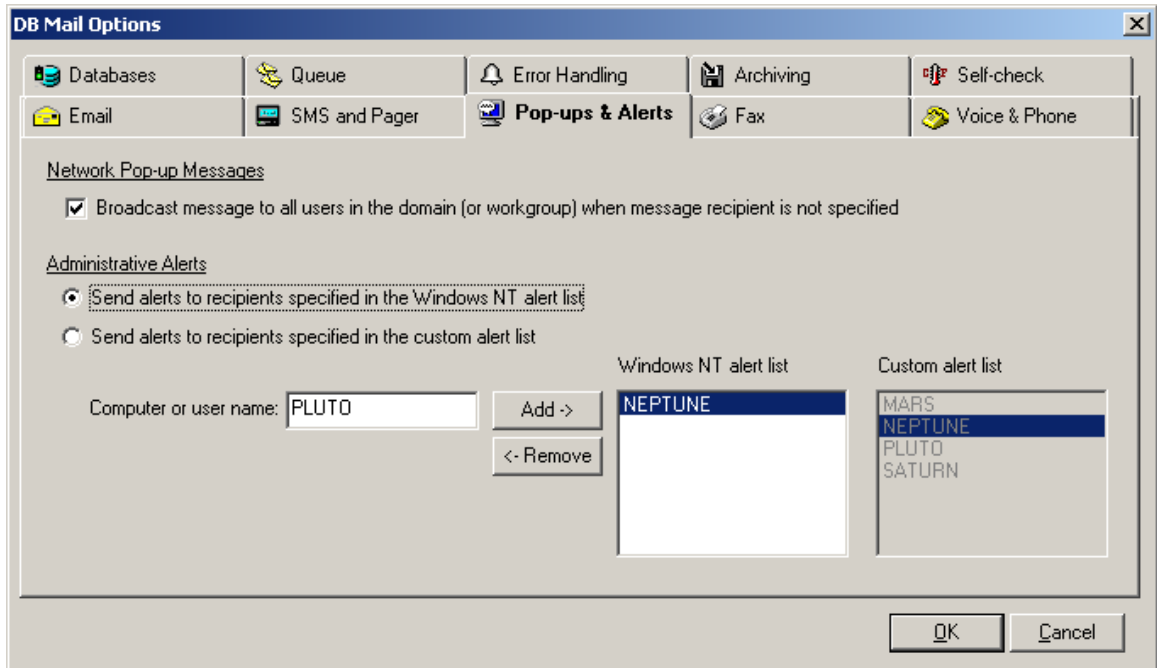


Configuration Option	Description
SNPP (pager) server	SNPP (pager) server – Name of your Simple Network Page Protocol server computer – just as with SMTP server this could be a network name or an IP address. This is the system name of your SNPP service provider (Telecom company). For example, Verizon uses snpp.myairmail.com. To find out your SNPP server name contact your service provider.
Port	Port – Port on which SNPP server listens for pager messages. Standard port number is 444 but different providers for security reasons may use different port numbers. Contact your service provider for more information.
Max. message size	Max. message size – maximum size of alpha (e.g. text) messages supported by your service provider and pager -- usually 240 characters.

Configuring Network Popups and Alerts Options

DB Mail can be configured to send network popup messages and alerts to computers/users in the same domain/workgroup. In Oracle, the Network Popups and Alerts interface is accessed via the SEND_POPUP_MESSAGE and SEND_ALERT function calls in the DB_MAIL package. In other databases the interface is accessed via the SendPopupMessage and SendAlert stored procedures or by directly inserting data into DBMAIL.PIPE table.

These functions require that NT Messenger service is running on the recipient's computers. Recipients must be running Windows NT 4, 2000, XP, 2003 or later. This functionality is identical to the Windows NT NET SEND command.



Configuration Option	Description
Broadcast option	Broadcast option – controls what DB Mail does when a NULL value is specified for the recipient parameter in the DB_MAIL.SEND_POPUP_MESSAGE function. If checked the message is displayed on every computer in the domain. If there is no domain (domain-less environment such as a peer-to-peer network), it is displayed on every computer in the workgroup. If this options is not checked messages with null recipient names are ignored.
Send alerts for recipients in NT alert list	Sends alerts for recipients in the NT alert list – controls how DB Mail executes DB_MAIL.SEND_ALERT function. If checked DB Mail will send interruptible alert messages to system administrators whose names are specified on the NT server.
Send alerts for recipients in the custom list	Sends alerts for recipients in the custom list – Allows an alternative list for use with DB_MAIL.SEND_ALERT. This may be helpful if the person who configures the DB Mail does not have admin rights or access to the alert list on NT server.
Computer or user name	Use this field to add specific computers or user names who should receive the pop-ups and alerts

Configuring Fax Options

DB Mail can be configured to send electronic faxes. In Oracle, the Fax interface is accessed via the SEND_FAX and SEND_FAX_EX function calls in the DB_MAIL package. In other databases the Fax interface is accessed via the SendFax and SendFaxEx stored procedures.

DB Mail Options

Databases Queue Error Handling Archiving Self-check
 Email SMS & Pager Pop-ups & Alerts Fax Voice & Phone

Fax Server Options

Fax queue folder: C:\Documents and Settings\All Users\Application Data\Microsoft\Windows NT\MSFax ...

Number of retries^{**}: 3 Retry interval (minutes): 10 ** Number of fax retries if the destination fax is busy or there is no answer.

Fax driver: Windows NT Fax Driver Fax printer: Fax Port: MSFAX

Fax servers^{*}: server1, server2, server3 * Enter comma-separated list of computer names or IP addresses.


Fax Cover Page

Default cover page: GENERIC

Company name: My Company

Mail address: 100 My Street State, Postal Code Telephone: (123) 456-7890 Fax: (123) 456-7891

OK Cancel

Configuration Option	Description
Fax queue folder	Folder where pending documents are queued for faxing. This folder is used by the Fax driver. Do not select DB Mail queue folder as it has nothing to do with DB Mail message queuing options.
Number of retries	This parameter controls how many times to retry faxing if the destination number is busy or not responding properly. If all attempts fail, the message is marked as failed and the document is moved to the Failed Faxes folder.
Retry interval	Minimum time to wait between fax retries.
Fax driver	Name of the fax driver as it appears in Windows Control Panel Printer's applet.
Fax servers	<p>Location of the fax server computer. The Fax server can be run on a computer where DB Mail is installed or any other computer on the network. DB Mail uses the fax server to process Fax messages. DB Mail actually prints fax messages to a Fax compatible graphic file, which it sends to the fax server for further processing.</p> <p> Tip: Fax transmissions are relatively slow operations. DB Mail provides built-in support for fax processing scaling out strategy (scaling out is the strategy that increases the capacity of an infrastructure tier to handle load by adding servers, thereby increasing the aggregate capacity of those servers). DB Mail server can be configured to work with multiple fax servers concurrently in order to increase the overall system throughput. As your fax processing volume grows you can add additional fax servers as needed and configure DB Mail server accordingly.</p> <p>To instruct DB Mail server to use multiple fax servers enter fax server computer names or IP addresses as a comma-separated list. Order of server names in the list is not important because DB Mail uses simple</p>

	round-robin method to communicate to multiple servers.
Fax Printer	Fax Printer name as it appears in Windows Control Panel Printer's applet.
Port	Port as it appears in Windows Control Panel Printer's applet.
Default cover page	Name of cover page file to use when NULL is specified for cover page parameter in DB_MAIL.SEND_FAX and DB_MAIL.SEND_FAX_EX functions. You can design cover pages using MS Cover Page Designer, which is a part of the Fax server software. Cover pages must be stored on the Fax server computer. For more information see Creating and modifying cover pages section.
Company name, mail address, phone, fax	Default values for Company name, mail address, phone, fax to be inserted into the specified cover page when DB_MAIL.SEND_FAX is used.

**Important Note:**

Fax Server software is installed by default on all Windows 2000/XP/2003 computers. Fax Server can run on any server or workstation. The computer must have a modem or modem pool and be connected to a phone line in order to send electronic faxes.

Configuring Voice Messaging Options

DB Mail can be configured to make phone calls and send pre-recorded or dynamically synthesized voice messages. In Oracle, the Voice Messaging interface is accessed via the SEND_VOICE function call in the DB_MAIL package. In other databases the Voice Messaging interface is accessed via the SendVoice stored procedure or by directly inserting data into DBMAIL.PIPE table.

DB Mail Options

Databases Queue Error Handling Archiving Self-check
 Email SMS & Pager Pop-ups & Alerts Fax **Voice & Phone**

Voice Message Server Options

Number of retries^{**}: Retry interval (minutes):


Voice servers^{*}:

Port:

Logon user:

^{**} Number of call retries if the destination number is busy or there is no answer. Retry parameters must be configured directly in the voice server properties.
^{*} Enter comma-separated list of computer names or IP addresses.

OK Cancel

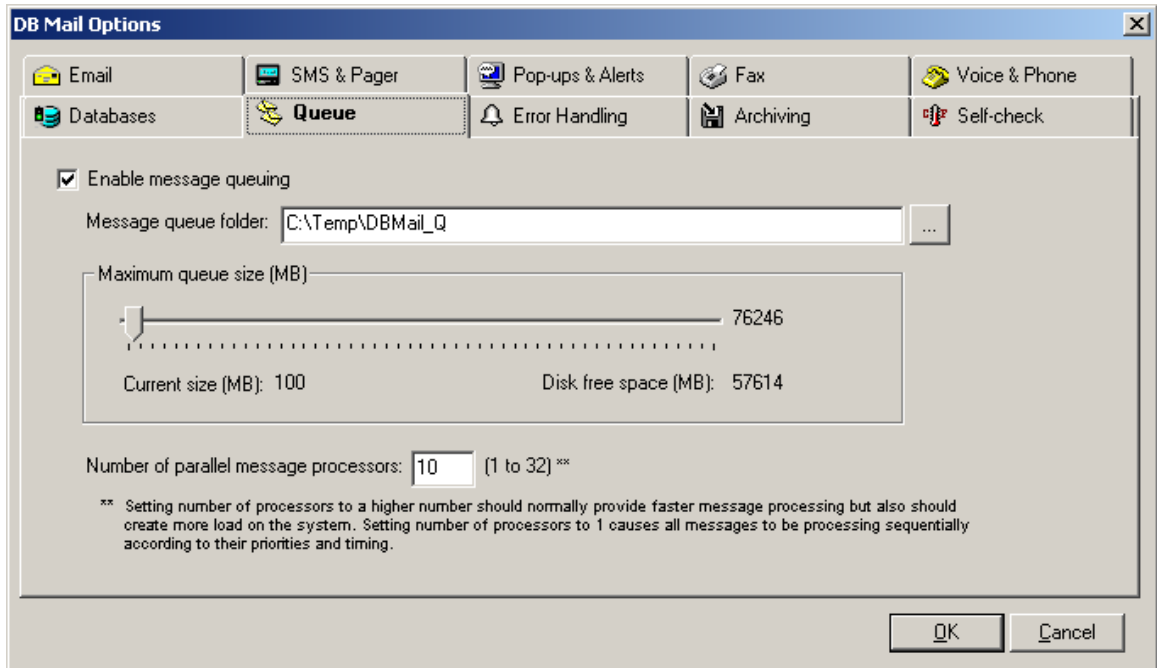
Configuration Option	Description
Number of retries	This parameter cannot be controlled through DB Mail interface and must be configured on the VoMS server using VoMS Administrator utility. This parameter controls how many times to retry calling if the destination number is busy or not responding properly. If all attempts fail, the message is marked as failed and the document is moved to the Failed Messages folder.
Retry interval	Minimum time to wait between phone call retries.
Voice servers	<p>Location of the VoMS server computer. The VoMS server can be run on a computer where DB Mail is installed or any other computer on the network. DB Mail uses the VoMS server to process voice messages. DB Mail actually uses the built-in VoMS client software to transmit messages to the to the VoMS server for further processing.</p> <p> Tip: Making phone calls and sending voice messages are relatively slow operations. DB Mail provides built-in support for voice processing scaling out strategy (scaling out is the strategy that increases the capacity of an infrastructure tier to handle load by adding servers, thereby increasing the aggregate capacity of those servers). DB Mail server can be configured to work with multiple VoMS servers concurrently in order to increase the overall system throughput. As your call processing volume grows you can add additional VoMS servers as needed and configure DB Mail server accordingly.</p> <p>To instruct DB Mail server to use multiple VoMS servers enter VoMS server computer names or IP addresses as a comma-separated list. Order of server names in the list is not important because DB Mail uses simple round-robin method to communicate to multiple servers.</p>
Port	Port number as it is configured in the VoMS server properties.
Logon user name	Name of the user for VoMS server authentication. This property cannot be changed and always defaults to DB MAIL.

 **Important Note:**

VoMS Server software is not installed by default. To install the limited version of the VoMS server that comes with every DB Mail license use the DB Mail setup program and choose to install VoMS Voice Message Server software. To install the full version contact SoftTree Technologies to obtain the VoMS server license. The VoMS server can be installed and can run on any server or workstation. The computer must have a sound card and a voice modem or Intel Dialogic phone board and be connected to a phone line in order to make phone calls and send voice messages.

Configuring Queue Options

For improved database performance DB Mail can queue messages obtained from the DB_MAIL database pipe. This asynchronous out-of-database message queuing allows DB Mail to obtain new messages while sending messages that arrived earlier. The new messages are extracted from the pipe almost immediately as they are placed to the DB_MAIL pipe by your SQL code running within the database. Therefore, your SQL code does not wait for the messages to be sent.



For disk space usage and processing efficiency, DB Mail stores messages in compressed format.

Configuration Option	Description
Enable message queuing	The Message Queue is optional. You can turn it on/off by selecting/deselecting this option.
Message queue folder	This is the directory where DB Mail stores queued messages and other temporary files. You can type the folder name or use the Browse button to select a different folder.
Maximum queue size	Specifies how much disk space to use for the files in the Message Queue folder. The more disk space you allot to the folder, the more messages DB Mail can store on your hard disk. If you are low on disk space, you might want to set this option to a lower percentage. If there is not enough space to store a newly arrived message, DB Mail stops processing new messages until free space become available in the folder. Note that sent messages are removed from the queue immediately so the space can be reused by other messages. To change the "maximum queue size" value, you can drag the slider control by mouse or for more precision positioning use Arrow Left and Arrow Right keys on your keyboard to move the slider control.



Important notes:

You must restart DB Mail Server before new changes for the Message Queue will take effect. If you change Message Queue folder and restart DB Mail while some messages are still waiting in the queue, all unsent messages will remain in the old message queue folder.

Configuring Error-Handling Options

For your convenience, DB Mail supports customizable error handling so that you can tune DB Mail to your error handling requirements.

Note that both email as well as Network pop-up messages can be sent for most Error conditions.

Configuration Option	Description
Database Communication Errors	This option instructs DB Mail to notify system administrators in case the database connection is broken, server is down or not available and so on.
Database Pipe Errors or Malformed Messages	This option instructs DB Mail to notify system administrators and the email sender in case DB Mail is unable to retrieve valid message data.
Email System Communication Errors	This option instructs DB Mail to notify system administrators in case it is unable to communicate to your email server. This option is applicable for SMTP email interface only. If you are using MAPI or Lotus Notes interface, use your email client options to configure error handling.
Email Delivery Errors	This option instructs DB Mail to notify system administrators and email sender in case your email server (SMTP interface) or your email client (MAPI and Lotus Notes) are returning an error indicating that the email message cannot be delivered. For example, this error can be detected if your email server does not recognize the recipient's address or sender's name. This error WILL NOT be detected if your MAPI or Lotus Notes client does accept the message but later (after the fact the message was sent by DB Mail) the email server rejects it. In that case, the email sender should receive an automated reply notice directly from the email server.
Administrator's email	This is the system administrator email address that DB Mail can use

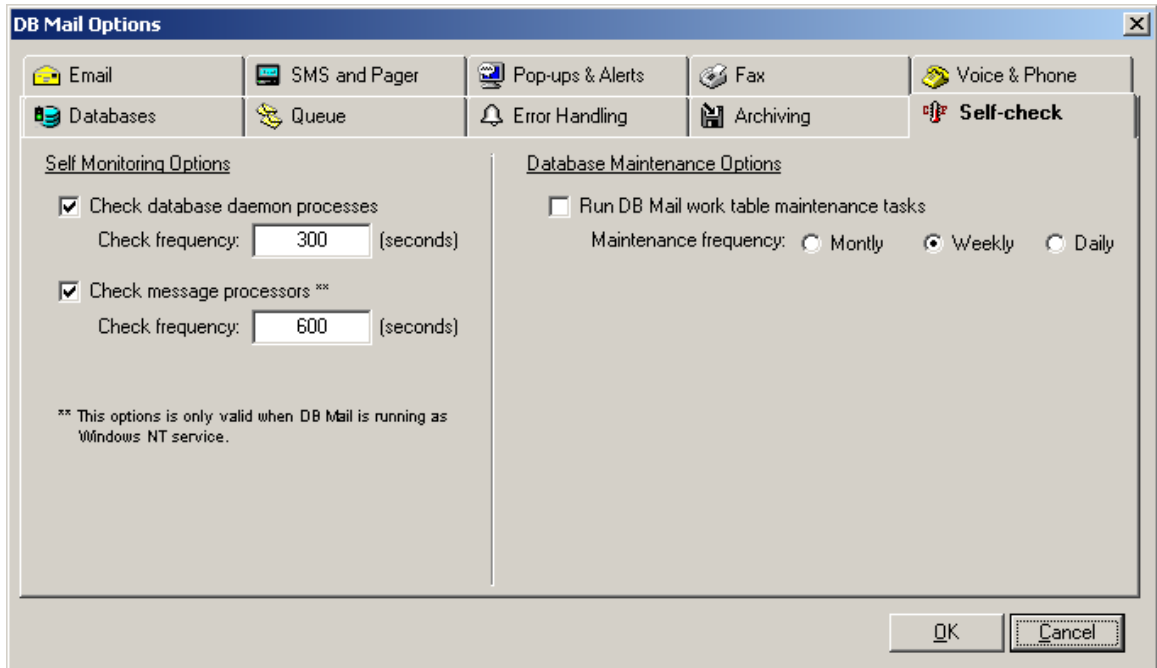
address	to send error notification messages. If you have more than one administrator whom you want to be notified in case of detected problems, specify their email addresses as a comma-separated list. If no email is specified, DB Mail ignores error-handling options for which it is supposed to send an email notification.
Administrator's network name	This is the system administrator network name that DB Mail can use to send error notification messages. If you have more than one administrator whom you want to be notified in case of detected problems, specify their network workgroup name so that everyone in the workgroup can be notified. If no network name is specified, DB Mail ignores error-handling options for which it is supposed to send a network message notification.
Recovery Options for Broken Database Connections	<p>Number of Retries – This option instructs DB Mail how many times it should attempt to reestablish broken database connections.</p> <p>Retry Interval – This option instructs DB Mail how many seconds it should wait before making a new attempt to reestablish a broken database connection.</p>
Recovery Options for Broken SMTP Communications	<p>Number of Retries – This option instructs DB Mail how many times it should attempt to resend an email message in case the email-server is not responding. This option applies to SMTP email interface only.</p> <p>Retry Interval – This option instructs DB Mail how many seconds it should wait before making a new attempt to resend an email message. This option applies to SMTP email interface only.</p>

**Important note:**

You must restart DB Mail Server before new changes for the Error Handling will take effect

Configuring Self-Healing and Maintenance Options

DB Mail supports the following self-monitoring and healing options



Configuration Option	Description
Check database daemon processes	<p>This option instructs DB Mail to monitor the internal database daemon processes and automatically restart them in case if database daemon process dies as a result of a hung or broken database connection.</p> <p>Note: DB Mail database daemons use internal "heart bit" procedures to periodically report to the DB Mail console their status. DB Mail console then verifies the reported values and if necessary restarts hung or failed daemons.</p> <p>Check frequency– This option controls how often DB Mail checks daemon statuses.</p>
Check message processors	<p>When running as service DB Mail service control process can periodically validate status of the DB Mail console and internal message queue processors.</p> <p>This option instructs DB Mail service control process to monitor the console and message queue processors and automatically restart them in case if they fail.</p> <p>Note: DB Mail database console uses internal "heart bit" procedures to periodically report to the DB Mail service control process statuses of the message queue processors. If a problem detected DB Mail service control process then automatically restarts the console and effectively all message queue processors.</p> <p>Check frequency– This option controls how often DB Mail checks message queue processor statuses.</p>
Database Maintenance Options	<p>Run DB Mail work table maintenance tasks – This option instructs DB Mail to periodically verify contents of the MAIL_ATTACH work table and purge data left as a result of old undeliverable messages.</p> <p>Monthly, weekly, daily – This option instructs DB Mail how often to</p>

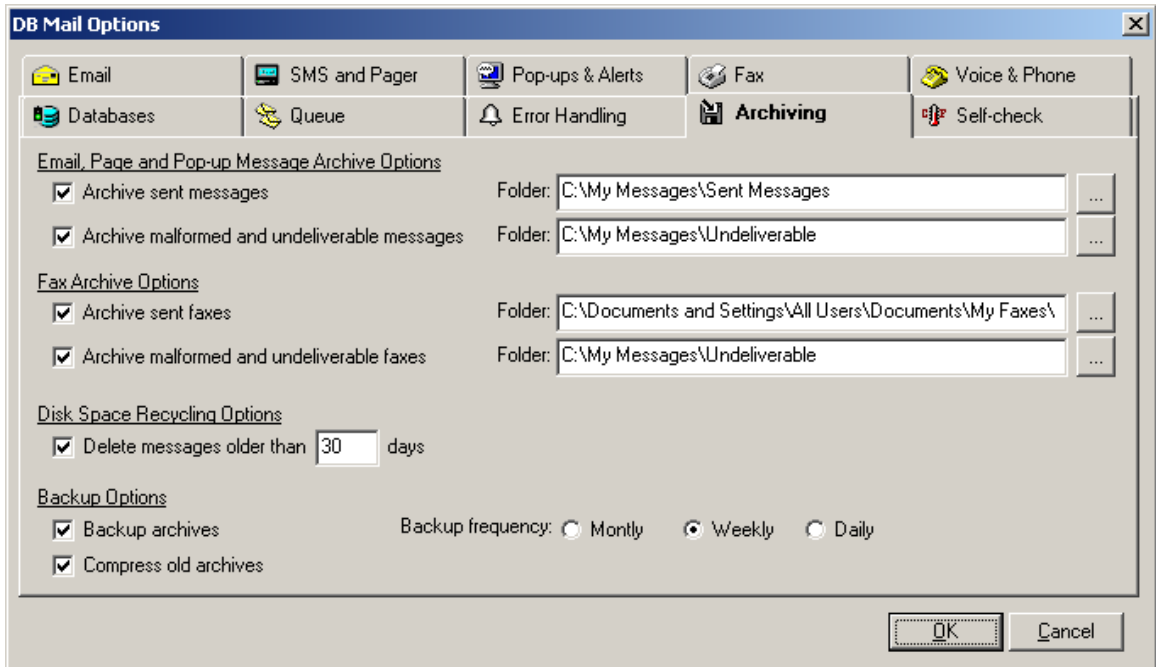
run the work table maintenance procedure.

 **Important note:**

You must restart DB Mail Server before new changes for the Self-Healing and Maintenance Options will take effect

Configuring Archiving Options

DB Mail can optionally log and archive all sent email messages. If archiving is enabled, every message that was successfully sent is appended to the MESSAGE.LOG file in the 'Sent Messages' folder. All undeliverable messages are appended to the MESSAGE.LOG file in the 'Undeliverable' folder. If archive backup option is enabled, DB Mail automatically renames these log files at the specified time and creates new log files as soon as it processes the next email message.



DB Mail Options

Email
 SMS and Pager
 Pop-ups & Alerts
 Fax
 Voice & Phone

Databases
 Queue
 Error Handling
 Archiving
 Self-check

Email, Page and Pop-up Message Archive Options

Archive sent messages Folder: C:\My Messages\Sent Messages

Archive malformed and undeliverable messages Folder: C:\My Messages\Undeliverable

Fax Archive Options

Archive sent faxes Folder: C:\Documents and Settings\All Users\Documents\My Faxes\

Archive malformed and undeliverable faxes Folder: C:\My Messages\Undeliverable

Disk Space Recycling Options

Delete messages older than days

Backup Options

Backup archives Backup frequency: Monthly Weekly Daily

Compress old archives

OK Cancel

Configuration Option	Description
Archive Sent Messages	This option instructs DB Mail to archive all messages sent successfully. Folder option – this is the folder in which DB Mail logs sent messages. You can type the folder name or use the Browse button to select a different folder.
Archive malformed and Undeliverable Messages	This option instructs DB Mail to archive all failed (in other words, undeliverable) messages as well as malformed messages. Folder option – this is the folder in which DB Mail logs undeliverable

	and malformed messages. You can type the folder name or use the Browse button to select a different folder.
Archive Sent Faxes	This option instructs DB Mail to archive all faxes sent successfully. Folder option – this is the folder in which DB Mail logs sent faxes. You can type the folder name or use the Browse button to select a different folder.
Archive malformed and Undeliverable Faxes	This option instructs DB Mail to archive all failed (in other words, undeliverable) faxes as well as malformed messages. Folder option – this is the folder in which DB Mail logs undeliverable and malformed faxes. You can type the folder name or use the Browse button to select a different folder.
Disk space recycling Options	This options allows space recycling by deleting all messages older than the specified number of days
Backup Archives	This option instructs DB Mail to periodically archive message logs. Archived logs are stored in the same directories where they were created. Depending on the value of the Compress Old Archives options, logs are either renamed and stored as .LOG files or renamed and compressed into standard .ZIP files. The name of the archived file is <i>messageMMDDYYYY .log</i> where <i>MMDDYYYY</i> is substituted with month, day and year when the archive was created.
Compress Old Archives	This option instructs DB Mail to stored archived logs as standard .ZIP files.
Backup Frequency	This option instructs DB Mail when to archive current logs and start new ones.

**Important note:**

You must restart DB Mail Server before new changes for the Archive Options will take effect.

Configuring User-Access and Security

Run DB Mail Database Setup program and then use Manage User Access option to grant/revoke user access. For more information see [Managing user access to DB Mail features](#) topic from the DB Mail installation chapter.

CHAPTER 6, Sending email messages

Overview

To send email messages from database applications use the `dbmail.SendMail` method or the `DB_MAIL.SEND_MAIL` call for Oracle. This method can be used to send email messages to single or multiple message recipients.

Messages can be sent in different email formats including TEXT, HTML, RTF, XML and other.

Email messages can include optional email attachments. If supported by your database attachments can be created from external files and/or from text and binary data stored in database tables. To send email message with attachments you must first load attachments using the `dbmail.AttachFile` or `dbmail.AttachData` methods. Text type attachments can be also created and deleted dynamically using the `dbmail.CreateMailFile` or `dbmail.DeleteMailFile` methods. In all databases except Sybase, the `dbmail.CreateMailFile` method creates attachment as operation system files on the database server computer. In Sybase this method creates attachments as chunks of text saved in the `dbmail.mail_file` table.

For detailed method descriptions and usage examples specific to your database system refer to the following topics in this chapter.

Oracle


SEND_MAIL

Use this method to send email messages from your database. The definition of SEND_MAIL function is shown below:

Definition

```
db_mail.send_mail(
    recipients      VARCHAR2,
    subject         VARCHAR2,
    message         VARCHAR2,
    reply_to       VARCHAR2      DEFAULT NULL,
    content_type    VARCHAR2      DEFAULT 'text/plain',
    priority        INTEGER        DEFAULT 1,
    attachment_id  NUMBER          DEFAULT NULL)
RETURN NUMBER
```

Argument	Max Size; Value Range	Description
Recipients	4000	Email address of message recipient. If you need to send the same message to multiple recipients, use comma to separate multiple recipient addresses. You can also specify valid email user groups. Email groups

		(also known as D-Lists or Distribution Lists) can be configured using your mail server administration tools.
Subject	255	Email message subject
Message	4000	Email message text.  Tip: If you need to send a message whose text is longer than the maximum size of varchar data-type in your database you can use a special attachment file with the name <i>message</i> and no extension. DB Mail will use the contents of that file for the message text. The value of the Message parameter will be ignored in that case. For more information see Error! Not a valid result for table. topic.
Reply-to (optional)	255	Sender's email address. Note that this property is used with SMTP interface only. Your server must support relay forwarding if you want to set the sender's email address to a value different from the one specified in the DB Mail email configuration. When using MAPI and Lotus Notes interfaces, DB Mail ignores this property. Also, note that for both MAPI and Lotus Notes, your email messages will have sender's address specified in the profile of the user whose account you used to configure DB Mail Server mail options.
Content_Type (optional)	50	Message content type is one of text/plain , text/html , text/xml . Note that this property is used with SMTP interface only. When using MAPI and Lotus Notes interfaces, DB Mail ignores this property. If you specify NULL or omit this parameter, DB Mail uses the default value, which is text/plain .
Priority (optional)	0..2	Message processing priority takes a value of 0 , 1 , or 2 . Note that this property is used only when Message Queuing is enabled in DB Mail options. Messages having higher priority numbers are processed before messages having lower priority numbers. Email messages sent using SMTP protocol also inherit this priority attribute.
Attachment_ID (optional)		ID of the record or group of records in the MAIL_ATTACH table containing attachment data. Message attachment should be created using <i>DB_MAIL.ATTACH_FILE</i> and <i>DB_MAIL.ATTACH_DATA</i> methods.

Return values:

0 - Success.

1 - Timed out. This procedure can timeout either because it cannot get a lock on the pipe, or because the pipe remains too full to be used.

3 - An interrupt occurred.

 **Usage Tips:**

If you add a call to the SEND_MAIL function from a multi-row SQL statement such as SELECT ... FROM ... TABLE, Oracle will invoke the SEND_MAIL function as many times as many rows are affected by the statement.

The following examples demonstrate how to send email messages using DB Mail functions.

Example 1 (SQL):

The following SELECT statement will send email to all users who run batch jobs and whose database password is going to expire next Monday. This assumes that the BATCH_USER table contains the EXPIRE_DATE, the USERID and FNAME columns, which represent the expiry date of the password, the email user id and the full user name.

```
SELECT db_mail.send_mail(userid || '@domain.com',
    'Password expiration',
    'Dear ' || fname || ', ' || chr(10) || chr(10) ||
    'Your database password will expire next Monday. ' ||
    'Be sure to update your batch jobs before that date.' || chr(10)
    || chr(10) ||
    'If you need assistance, reply to this message with ' ||
    'your questions',
    'helpdesk@domain.com' )
FROM batch_user
WHERE expire_date = TRUNC(SYSDATE + 3);
```

Example 2 (PL/SQL):

This is a more advanced PL/SQL example of a similar use of the DB_MAIL.SEND_MAIL function:

```
/******
 * This functions debits the specified account by specified amount
 * only if there are sufficient funds to cover the withdrawal, and if there
 * are not, it sends an email alert to the account manager.
 * This function returns new account balance.
 *****/
CREATE OR REPLACE FUNCTION acct_debit (acct_nbr CHAR(10),
    debit_amt NUMBER(5, 2))

RETURN INTEGER
AS
DECLARE
    acct_balance NUMBER(11,2);
    ret_code INTEGER := -1;
BEGIN

    SELECT balance INTO acct_balance
    FROM accounts
    WHERE acct = acct_nbr
    FOR UPDATE OF balance;

    IF acct_balance >= debit_amt THEN
        BEGIN
            acct_balance := acct_balance - debit_amt
            UPDATE accounts
            SET balance = acct_balance
            WHERE acct = acct_nbr;

            COMMIT;

            ret_code := 1; -- success
        END;
    ELSE
        -- Insufficient funds.
```

```

-- Send email notification to the account manager

SELECT db_mail.send_mail(m.email,
                        'WARNING: Insufficient funds',
                        'Time: ' || SYSDATE || chr(10) ||
                        ' Account: ' || acct_nbr, NULL)
INTO ret_code
FROM managers m, accounts a
WHERE m.mgr_id = a.mgr_id
      AND a.acct = acct_nbr;

-- If email was sent successfully set ret_code to 0,
-- otherwise leave it as -1
IF ret_code = 1 THEN ret_code := 0; END IF;
END IF;

RETURN ret_code; -- return code 1 indicates success
-- 0 indicates insufficient funds and successful
-- notice
-- -1 indicates insufficient funds and failed
-- notice

END acct_debit;

```

Example 3 (sending email in HTML format):

The following SELECT statement will send the email to all users who run batch jobs and whose database password is going to expire next Monday. It is similar to the first example, but uses HTML rather than plain text – note the use of the 'text/html' content type description:

```

SELECT dbmail.send_mail(userid || '@domain.com',
                        'Password expiration',
                        '<p>Dear ' || fname || ',</p>' ||
                        '<p>Your database password will <b>expire</b> in 3 days. ' ||
                        'Be sure to update your batch jobs before that date.</p>' ||
                        '<p><font color="#008000">If you need assistance, ' ||
                        'reply to this message with your questions</font></p>',
                        'helpdesk@domain.com',
                        'text/html' )
FROM batch_user
WHERE expire_date = TRUNC(SYSDATE + 3);

```

ATTACH_FILE


Use this method to attach external files to email messages sent using SEND_MAIL function. To attach data already stored inside the database in BLOB columns, use the ATTACH_DATA function. The definition of ATTACH_FILE function is shown below:

Definition

```

FUNCTION db_mail.attach_file(
        id                NUMBER,
        file_name         VARCHAR2,
        dir               VARCHAR2 )
RETURN NUMBER

```

Argument	Max size; Value range	Description
ID		Attachment group ID. In order to generate unique Attachment IDs, pass NULL as a parameter when calling ATTACH_DATA or ATTACH_FILE for the first time for every email message. The called function will return a unique ID that you must use in subsequent calls when attaching more files or data to the same message.
File_Name	255	Name of the attached file as it will appear in the email message. The name must conform to standard file naming conventions and should not include file path. File name must be unique within a single email message.
Dir	30	Name of Oracle directory object pointing to the existing operation system directory on the database server computer where the specified File_Name can be found.  IMPORTANT: Do not confuse directory object name with the name of the actual directory containing files.

Return values:

-1 - Error. File cannot be opened or cannot be read. Verify file name and directory name are correct. Also check that File_Name is not NULL.

Other – New attachment group ID is returned if NULL value is passed for the **ID** argument, otherwise the value of the **ID** argument is returned.

 **Usage Tips:**

- If you want to attach more than one external file, call the ATTACH_FILE function as many times as many files you want to add.
- To attach data already stored inside the database in BLOB columns, use the ATTACH_DATA function.
- To attach both external files and data, call both functions as many times as there are attachments you want to add. Note that every attachment will appear as a file in the resulting email message. Names of files in the resulting message will match the names of files that you have specified as arguments for the functions.
- In order to attach files you must have an Oracle DIRECTORY object created. You use that object to specify where to look for the files to be attached. If you have files in multiple directories, create one DIRECTORY object for each of them.

To create a DIRECTORY object in an Oracle 8, 8i or 9i database use Oracle CREATE DIRECTORY SQL statement, which can be executed from SQL*Plus or other SQL Editor.

Syntax:

```
CREATE [OR REPLACE] DIRECTORY directory AS 'pathname';
```

Examples:

In Windows systems: `CREATE DIRECTORY images AS 'c:\myfiles\images';`

In UNIX systems: `CREATE DIRECTORY images AS '/public/LOB/files/images';`

- If you want other users to be able to attach your files, make sure to grant them necessary read permissions for your directory object.

Example:

To grant READ on directory images to user scott, execute the following statement:

```
GRANT READ ON DIRECTORY images TO scott;
```

- Calls to `ATTACH_FILE` and `ATTACH_DATA` functions can be nested, for example `ATTACH_FILE(ATTACH_FILE(NULL, 'file1.doc', 'MY_FILES'), 'file2.doc', 'MY_FILES')`
- **Oracle 7 limitations:** Mail attachments are not supported with Oracle 7.

**Important notes:**

- DB Mail temporarily stores all attachments in the `MAIL_ATTACH` table. One message can have up to 255 attachments. Every message must have a unique Attachment ID referencing attachments that belong to that message. Within this group of records referenced by the Attachment ID, every individual attachment must have a unique File ID and File Name. Both Attachment ID and File ID are used as the surrogate key.
- DB Mail automatically deletes all attachments from the `MAIL_ATTACH` table after the message is sent or queued for sending.
- You must create separate attachments for every email message. To send an identical message to multiple recipients either send it to a user group using group name as recipient address or specify all recipients as a comma-separated list. The maximum length of the list is limited by the size of `VARCHAR` variable in your Oracle database version (normally 4000 characters). Only one copy of attachments is needed in both cases. In all other cases you must create separate messages with separate attachments for every recipient.

Examples

The following examples demonstrate how to send email messages with attachments using DB Mail functions. Here are six brief examples for attaching an external file and data from BLOB columns. Note that each example is specific to certain Oracle versions and attachment types.

Example 1 (Oracle 8, 8i, 9i, and 10g, BFILE, 1 message, 1 attachment):

```
DECLARE
  attach_id INTEGER;
  ret_code INTEGER;
BEGIN
  attach_id := db_mail.Attach_File(NULL, 'product.gif', 'IMAGES');

  ret_code := db_mail.send_mail('user@domain.com',
    'Test message with attachments',
    'This message has one attachment',
    NULL,
```

```

        NULL,
        NULL,
        attach_id) ;
END;
```

Example 2 (Oracle 8, 8i, 9i, and 10g, BFILE, 1 message, 3 attachments):

```

DECLARE
    attach_id INTEGER;
    ret_code INTEGER;
BEGIN
    attach_id := db_mail.Attach_File(NULL, 'product1.gif', 'IMAGES');
    ret_code := db_mail.Attach_File(attach_id, 'product2.gif', 'IMAGES');
    ret_code := db_mail.Attach_File(attach_id, 'products3.gif', 'IMAGES');

    ret_code := db_mail.send_mail('user@domain.com',
        'Test message with attachments',
        'This message has three attachments',
        NULL,
        NULL,
        NULL,
        attach_id) ;
END;
```

Example 3 (Oracle 8, 8i, 9i, and 10g, BLOB, 1 message, 1 attachment):

```

DECLARE
    attach_id INTEGER;
    ret_code INTEGER;
    src_data BLOB;
BEGIN
    SELECT b_col
    INTO src_data
    FROM lob_table
    WHERE key_value = 21;

    attach_id := db_mail.Attach_Data(NULL, 1, 'product.gif', src_data);

    ret_code := db_mail.send_mail('user@domain.com',
        'Test message with attachments',
        'This message has one attachment',
        NULL,
        NULL,
        NULL,
        attach_id) ;
END;
```

Example 4 (Oracle 8, 8i, 9i, and 10g, BLOB, 1 message, 3 attachments):

```

DECLARE
    attach_id INTEGER;
    ret_code INTEGER;
    src_data BLOB;
BEGIN
    SELECT b_col
    INTO src_data
    FROM lob_table
    WHERE file_name = 'product1.gif';
```

```

attach_id := db_mail.Attach_Data(NULL, 1, 'product1.gif', src_data);

SELECT b_col
INTO src_data
FROM lob_table
WHERE file_name = 'product2.gif';

ret_code := db_mail.Attach_Data(attach_id, 2, 'product2.gif', src_data);

SELECT b_col
INTO src_data
FROM lob_table
WHERE file_name = 'product3.gif';

ret_code := db_mail.Attach_Data(attach_id, 3, 'product3.gif', src_data);

ret_code := db_mail.send_mail('user@domain.com',
    'Test message with attachments',
    'This message has three attachments',
    NULL,
    NULL,
    NULL,
    attach_id) ;
END;

```

Example 5 (Oracle 8i, 9i and 10g, BFILE, many messages, 1 attachment):

```

SELECT db_mail.send_mail(customer_email,
    'New great product in our store',
    'We are happy to offer the product that you have asked for.' ||
    'For details please see the attached picture.',
    NULL,
    NULL,
    NULL,
    db_mail.Attach_File(NULL, 'product.bmp', 'IMAGES')) ;
FROM customer
WHERE customer_email IS NOT NULL
    AND status = 'A';

```

Example 6 (Oracle 8i, 9i and 10g, BFILE, many messages, 2 attachments):

```

SELECT db_mail.send_mail(customer_email,
    'New great products in our store',
    'We are happy to offer two product that you have asked for. ' ||
    'For details please see the attached pictures.',
    NULL,
    NULL,
    NULL,
    db_mail.Attach_File(Attach_File(NULL, 'product1.bmp', 'IMAGES'),
        'product2.bmp', 'IMAGES')) ;
FROM customer
WHERE customer_email IS NOT NULL
    AND status = 'A';

```

ATTACH_DATA

Use this method to attach data already stored inside the database in BLOB columns to email messages sent using SEND_MAIL function. To attach external files, use the ATTACH_FILE function. The definition of ATTACH_DATA function is shown below:

Definition

```

FUNCTION db_mail.attach_data(
            id          NUMBER,
            file_name   VARCHAR2,
            data        BLOB )
RETURN NUMBER

```

Argument	Max size; Value range	Description
ID		Attachment group ID. In order to generate unique Attachment IDs, pass NULL as a parameter when calling ATTACH_DATA or ATTACH_FILE for the first time for every email message. The called function will return a unique ID that you must use in subsequent calls when attaching more files or data to the same message.
File_Name	255	Name of the attached file as it will appear in the email message. The name must conform to standard file naming conventions and should not include file path. File name must be unique within single email message.
Data	2GB	BLOB value (BLOB, LOB, CLOB, etc) containing the actual attachment data.

Return values:

-1 - Error. Invalid blob value or invalid parameters. Check that File_Name is not NULL. Also check BLOB value is initialized and the referenced BLOB column is not locked.

Other – New attachment group ID is returned if NULL value is passed for the **ID** argument, otherwise the value of the **ID** argument is returned.

 **Usage Tips:**

- If you want to attach more than one BLOB, call the ATTACH_DATA function as many times as many as the number of BLOB values you want to add.
- To attach external files, use the ATTACH_FILE function.
- To attach both external files and data, call both functions as many times as there are attachments you want to add. Note that every attachment will appear as a file in the resulting email message. Names of files in the resulting message will match names of files that you have specified as arguments for the functions.
- Calls to ATTACH_FILE and ATTACH_DATA functions can be nested, for example

```

ATTACH_DATA( ATTACH_DATA( NULL, 'file1.doc', var_blob1),
            'file2.doc', var_blob2)

```
- **Oracle 7 limitations:** Mail attachments are not supported with Oracle 7.

**Important notes:**

- DB Mail temporarily stores all attachments in the MAIL_ATTACH table. One message can have up to 255 attachments. Every message must have a unique Attachment ID referencing attachments that belong to that message. Within this group of records referenced by the Attachment ID, every individual attachment must have a unique File ID and File Name. Both Attachment ID and File ID are used as the surrogate key.
- DB Mail automatically deletes all attachments from the MAIL_ATTACH table after the message is sent or queued for sending.
- You must create separate attachments for every email message. To send an identical message to multiple recipients either send it to a user group using group name as recipient address or specify all recipients as a comma-separated list. The maximum length of the list is limited by the size of VARCHAR variable in your Oracle database version (normally 4000 characters). Only one copy of attachments is needed in both cases. In all other cases you must create separate messages with separate attachments for every recipient.

Examples

See examples for the `ATTACH_FILE` function that also includes examples for the `ATTACH_DATA`.

CREATE_MAIL_FILE

Use this method to write various flat files to be attached to email messages or used with DB Mail fax procedures. The definition of `CREATE_MAIL_FILE` procedure is shown below:

Definition

```
PROCEDURE db_mail.create_mail_file(
    file_name    VARCHAR2,
    text         VARCHAR2,
    append       BOOLEAN )
```

Argument	Max size; Value range	Description
File_Name	255	Name of the file to write to. File name should not include file path. The file will be created in the directory on the database server computer. The actual file location is specified by SYSTEM.MAILSTORE directory object.
Text	32765	Text to write to the file specified by the File_Name argument.
Append	TRUE or FALSE	Value of the Append argument specifies whether to create a new file or append value of the Text argument to an existing file. If a value of TRUE is passed for the Append argument and the file already exists, <code>CREATE_MAIL_FILE</code> overwrites the existing file. If a value of FALSE is passed for the Append argument and the file does not exist, <code>CREATE_MAIL_FILE</code> fails and an application error is raised.

Return values:

None. Oracle procedures do not return values.

If an error occurs one of the predetermined exceptions is raised. In case an exception is caught, check Oracle SQLERRM session variable to obtain the error description.

**Usage Tips:**

- Call CREATE_MAIL_FILE as many times for as many chunks of text you want to write to the file.
- When calling CREATE_MAIL_FILE for the first time to write the very first chunk of text, specify FALSE for the Append parameter. In subsequent calls specify TRUE.
- If you use CREATE_MAIL_FILE to write a file attachment for the ATTACH_FILE function or to write a fax file for the SEND_FAX or SEND_FAX_EX function and you do not need that file after calling one of the Send methods you can delete it using DELETE_MAIL_FILE procedure. Please note that DELETE_MAIL_FILE function is only available in Oracle 9i version and later. If you are running Oracle 8 or Oracle8i you cannot delete files from within a database procedure or function. Instead you should schedule a periodic operation system job to delete temporary files created with CREATE_MAIL. When deleting files be careful to not delete files that have been written already but not loaded yet. To avoid this timing issue, only delete files that are older than a few hours.

Examples

The following examples demonstrate how to use CREATE_MAIL_FILE and DELETE_MAIL_FILE auxiliary procedures.

Example 1 (create email attachment):

```

DECLARE
  ret_code INTEGER;
BEGIN
  db_mail.create_mail_file('report.htm',
    '<html><title>Test Report</title><body>' || chr(10) ||
    '<h2>Test Report</h2><hr>' || chr(10) ||
    '<table><tr><th bgcolor=black><font color=white>Col 1</th>' ||
    '<th bgcolor=black><font color=white>Col 2</th></tr>' || chr(10) ||
    '<tr><td>Value A</td><td align=right>1</td></tr>' || chr(10) ||
    '<tr><td>Value B</td><td align=right>2</td></tr>' || chr(10) ||
    '</table></body></html>', FALSE );

  ret_code := db_mail.send_mail('me@mycompany.com',
    'Test message with attachments',
    'The test report is attached. Please ignore this message.',
    'myname@mycompany.com',
    'text/plain',
    1,
    db_mail.attach_file(NULL, 1, 'report.htm', 'MAILSTORE'));

  IF ret_code != 0 THEN
    raise_application_error(-20010, 'SEND_MAIL Status = ' || ret_code);
  END IF;
END;
```

Example 2 (create fax document):

```

DECLARE
  ret_code INTEGER;
BEGIN
  db_mail.create_mail_file('report.htm',
    '<html><title>Test Report</title><body>' || chr(10) ||
    '<h2>Test Report</h2><hr>' || chr(10) ||
    '<table><tr><th bgcolor=black><font color=white>Col 1</th>' ||
    '<th bgcolor=black><font color=white>Col 2</th></tr>' || chr(10) ||
    '<tr><td>Value A</td><td align=right>1</td></tr>' || chr(10) ||
    '<tr><td>Value B</td><td align=right>2</td></tr>' || chr(10) ||
    '</table></body></html>', TRUE );

  ret_code := db_mail.send_fax('+1 (123) 222-3344',
    'Test fax subject',
    'report.htm',
    'Generic',
    'Test recipient');


  db_mail.delete_mail_file('report.htm');

  IF ret_code != 0 THEN
    raise_application_error(-20010, 'SEND_FAX Status = ' || ret_code);
  END IF;
END;

```

DELETE_MAIL_FILE

Use this method to delete files created using `CREATE_MAIL_FILE` procedure.

 **Note:** `DELETE_MAIL_FILE` is supported only in Oracle 9i and later.

Definition

```

PROCEDURE db_mail.delete_mail_file(
  file_name    VARCHAR2 )

```

Argument	Max size; Value range	Description
File_Name	255	Name of the file to delete. File name should not include file path. The file must exist on the database server computer. The actual file location is specified by <code>SYSTEM.MAILSTORE</code> directory object.

Return values:

None. Oracle procedures do not return values.

If an error occurs one of the predetermined exceptions is raised. In case if an exception is caught check Oracle `SQLERRM` session variable to obtain the error description.

 **Usage Tips:**

- If you use CREATE_MAIL_FILE to write a file attachment for the ATTACH_FILE function or to write a fax file for the SEND_FAX or SEND_FAX_EX functions and you do not need that file after calling one of the Send methods you can delete it using DELETE_MAIL_FILE procedure. Please note that DELETE_MAIL_FILE function is only available in Oracle 9i version and later. If you are running Oracle 8 or Oracle8i you cannot delete files from within a database procedure or function. Instead you should schedule a periodic operation system job to delete temporary files created with CREATE_MAIL. When deleting files be careful to not delete files that have been written already but not loaded yet. To avoid this timing issue delete only files that are older than a few hours.

Examples

See example 2 available in the [CREATE_MAIL_FILE](#) topic.


Microsoft SQL Server

SendMail

Use this method to send email messages from your database. The definition of SendMail function is shown below:

Definition

```
PROCEDURE dbmail.SendMail(
    @recipients    VARCHAR(8000),
    @subject       VARCHAR(255),
    @message       VARCHAR(8000),
    @reply_to      VARCHAR(255)= NULL,
    @content_type  VARCHAR(50) = 'text/plain',
    @priority      INT = 1,
    @attachment_id INT = NULL)
```

Argument	Max Size; Value Range	Description
Recipients	8000	Email address of message recipient. If you need to send the same message to multiple recipients, use comma to separate multiple recipient addresses. You can also specify valid email user groups. Email groups (also known as D-Lists or Distribution Lists) can be configured using your mail server administration tools.
Subject	255	Email message subject
Message	8000	Email message text.  Tip: If you need to send a message whose text is longer than the maximum size of varchar data-type in your database you can use a special attachment file with the name <i>message</i> and no extension. DB Mail will use the contents of that file for the message text. The value of the Message parameter will be ignored in that case. For more

		information see Error! Not a valid result for table. topic.
Reply-to (optional)	255	Sender's email address. Note that this property is used with SMTP interface only. Your server must support relay forwarding if you want to set the sender's email address to a value different from the one specified in the DB Mail email configuration. When using MAPI and Lotus Notes interfaces, DB Mail ignores this property. Also note that for both MAPI and Lotus Notes, your email messages will have sender's address specified in the profile of the user whose account you used to configure DB Mail Server mail options.
Content_Type (optional)	50	Message content type is one of text/plain , text/html , text/xml . Note that this property is used with SMTP interface only. When using MAPI and Lotus Notes interfaces, DB Mail ignores this property. If you specify NULL or omit this parameter, DB Mail uses the default value, which is text/plain .
Priority (optional)	0..2	Message processing takes a value of 0 , 1 , or 2 . Note that this property is used only when Message Queuing is enabled in DB Mail options. Messages having higher priority numbers are processed before messages having lower priority numbers. Email messages sent using SMTP protocol also inherit this priority attribute.
Attachment_ID (optional)		ID of the record or group of records in the MAIL_ATTACH table containing attachment data. Message attachment should be created using <i>AttachFile</i> and <i>AttachData</i> methods.

Return values: Returns unique message ID or returns -1 if an error occurs.



Usage Tips:

Use the @@ERROR global variable to check for errors. @@ERROR is set to 0 if the procedure executed successfully. If an error occurs, a non-zero number is returned. @@ERROR returns the number of the error message until another Transact-SQL statement is executed. You can view the text associated with an @@ERROR error number in the SYSMESSAGES system table.

Examples

The following examples demonstrate how to send email messages using DB Mail functions.

Example 1:

The following EXECUTE statement will send email to all users who run batch jobs and whose database password is going to expire next Monday. This assumes that the BATCH_USER table contains the EXPIRE_DATE, the USERID and FNAME columns, which represent the expiry date of the password, the Mail email user id and the full user name.

```
EXEC master.dbmail.SendMail 'batch_users@domain.com',
    'Password expiration',
    'Attention batch job owners, your database password will
     expire in 3 days. Be sure to update your batch jobs before
     that date. If you need assistance, reply to this message
     with your questions',
    'helpdesk@domain.com',
    'text/plain'
```

Example 2:

In this example we create user defined scalar function callable from various SQL statement just like other built-in system functions. Using this function we will send email to all users who run batch jobs and whose database passwords are going to expire next Monday. This example assumes that the BATCH_USER table contains the EXPIRE_DATE, the USERID and FNAME columns, which represent the expiry date of the password, the email user id and the full user name.

```
-- First, let's create a user-defined scalar Transact-SQL function that
-- we can call from SELECT statements
CREATE FUNCTION mySendMail( @recipient VARCHAR(30),
                           @subject VARCHAR(255),
                           @message VARCHAR(8000) )
RETURNS INT
AS
BEGIN
    DECLARE @ret INT
    EXEC @ret = master.dbmail.SendMail @recipient, @subject,
                                       @message, 'helpdesk@domain.com'

    RETURN (@ret)
END
go

-- Now, we can call our own send mail function
SELECT mySendMail(userid + '@domain.com',
                  'Password expiration',
                  'Dear ' + fname + ', ' + char(10) + char(10) +
                  'Your database password will expire next Monday. ' +
                  'Be sure to update your batch jobs before that date.' + char(10)
                  + char(10) +
                  'If you need assistance, reply to this message with ' +
                  'your questions' )
FROM batch_user
WHERE expire_date =
      DateAdd(DAY, 3, convert(datetime, convert(varchar, GetDate(), 101)))
go
```

Example 3:

This is a more advanced T-SQL example of a similar use of the dbmail.SendMail function:

```
/*
 * This procedure debits the specified account by specified amount
 * only if there are sufficient funds to cover the withdrawal, and if there
 * are not, it sends an email alert to the account manager.
 * This function returns new account balance.
 */
CREATE PROCEDURE sp_acct_debit (@acct_nbr CHAR(10), @debit_amt MONEY)
AS
BEGIN
    DECLARE
        @acct_balance MONEY,
        @ret_code INTEGER
    SET @ret_code = -1

    SELECT @acct_balance = balance
    FROM accounts
    WHERE acct = @acct_nbr
```

```

IF @acct_balance >= @debit_amt
BEGIN
    SET @acct_balance = @acct_balance - @debit_amt
    UPDATE accounts
    SET balance = @acct_balance
    WHERE acct = @acct_nbr

    IF @@error = 0 SET @ret_code = 1 -- success
END
ELSE
    -- Insufficient funds.
    -- Send email notification to the account manager
    DECLARE @email VARCHAR(50),
            @message VARCHAR(200)

    SELECT @email = m.email,
           @message = 'Time: ' + convert(varchar, GetDate()) + char(10) +
                    'Account: ' + convert(varchar, @acct_nbr)
    FROM managers m, accounts a
    WHERE m.mgr_id = a.mgr_id
          AND a.acct = @acct_nbr

    EXEC @rec_code = master.dbo.SendMail @email,
                                         'WARNING:Insufficient funds',
                                         @message

    -- If email was sent successfully set ret_code to 0,
    -- otherwise leave it as -1
    IF @ret_code = 1 SET @ret_code = 0 ELSE SET @retcode = -1
END

RETURN (@ret_code) -- return code 1 indicates success
                -- 0 indicates insufficient funds and
                --      successful notice
                -- -1 indicates insufficient funds and failed
                --      notice
END

```

Example 4 (sending email in HTML format):

The following EXECUTE statement will send emails to batch_users email group which includes users who run batch jobs. It uses HTML rather than plain text – note the use of the 'text/html' content type description:

```

EXEC master.dbo.SendMail 'batch_users@domain.com',
    'Password expiration',
    '<p>Attention batch job owners,</p>
    <p>Your database password will <b>expire</b> in 3 days.
    Be sure to update your batch jobs before that date.</p>
    <p><font color="#008000">If you need assistance,
    reply to this message with your questions</font></p>',
    'helpdesk@domain.com',
    'text/html'

```

AttachFile

Use this method to attach external files to email messages sent using dbmail.SendMail procedure. To

attach data already stored inside the database in TEXT and IMAGE columns, use the AttachData procedure. The definition of AttachFile procedure is shown below:

Definition

```
PROCEDURE dbmail.AttachFile(
    @id          INT = NULL,
    @file_name   VARCHAR(255) )
```

Argument	Max size; Value range	Description
ID		Attachment group ID. In order to generate unique Attachment IDs, pass NULL as a parameter when calling AttachData or AttachFile for the first time for every email message. The called procedure will return a unique ID that you must use in subsequent calls when attaching more files or data to the same message.
File_Name	255	Name of the attached file as it will appear in the email message. The name must conform to standard file naming conventions and may include file path. File name must be unique within single email message.

Return values: Returns unique attachment group ID or returns -1 if an error occurs.



Important Note: New attachment group ID is returned if NULL is passed for the **ID** argument, otherwise the value of the **ID** argument is returned.



Usage Tips:

- If you want to attach more than one external file, call the AttachFile procedure as many times as many files you want to add.
- To attach data already stored inside the database in TEXT and IMAGE columns, use the AttachData procedure.
- To attach both external files and data, call both procedures as many times as there are attachments you want to add. Note that every attachment will appear as a file in the resulting email message. Names of files in the resulting message will match names of files that you have specified as arguments for the procedures.



Important notes:

- DB Mail temporarily stores all attachments in the MAIL_ATTACH table. One message can have up to 255 attachments. Every message must have a unique Attachment ID referencing attachments that belong to that message. Within this group of records referenced by the Attachment ID, every individual attachment must have a unique File ID and File Name. Both Attachment ID and File ID are used as the surrogate key.
- DB Mail automatically deletes all attachments from the MAIL_ATTACH table after the message is sent or queued for sending.

- You must create separate attachments for every email message. To send an identical message to multiple recipients either send it to a user group using group name as recipient address or specify all recipients as a comma-separated list. The maximum length of the list is limited by the size of VARCHAR variable in your SQL Server version (normally 8000 characters). Only one copy of attachments is needed in both cases. In all other cases you must create separate messages with separate attachments for every recipient.

Examples

The following examples demonstrate how to send email messages with attachments using DB Mail procedures. Here are several brief examples for attaching an external file and data from IMAGE columns.

Example 1 (1 message, 1 file attachment):

```
DECLARE @attach_id INTEGER

EXEC @attach_id = master.dbmail.AttachFile NULL, 'c:\images\product.gif'

EXEC master.dbmail.SendMail 'user@domain.com',
    'Test message with attachments',
    'This message has one attachment',
    NULL,
    NULL,
    NULL,
    @attach_id
```

Example 2 (1 message, 3 file attachments):

```
DECLARE @attach_id INTEGER

EXEC @attach_id = master.dbmail.AttachFile NULL, 'c:\images\product1.gif'
EXEC master.dbmail.AttachFile @attach_id, 'c:\images\product2.gif'
EXEC master.dbmail.AttachFile @attach_id, 'c:\images\products3.gif'

EXEC master.dbmail.SendMail 'user@domain.com',
    'Test message with attachments',
    'This message has three attachments',
    NULL,
    NULL,
    NULL,
    @attach_id
```

Example 3 (1 message, 1 BLOB attachment):

```
DECLARE @attach_id INTEGER
DECLARE @data_length INT
DECLARE @ptrval BINARY(16)

-- read blob value from a table
SELECT @data_length = DATALENGTH(blob_column),
       @ptrval = TEXTPTR(blob_column)
FROM images_table
WHERE image_id = 21

READTEXT images_table.blob_column @ptrval 1 @data_length

-- pass that value to the AttachData procedure
```



```
EXECUTE @attach_id = master.dbmail.AttachData NULL, 'product.gif', @ptrval

-- now, mail it
EXEC master.dbmail.SendMail 'user@domain.com',
    'Test message with attachments',
    'This message has one attachment',
    NULL,
    NULL,
    NULL,
    @attach_id
```

Example 4 (1 message, 3 BLOB attachments):

```
DECLARE @attach_id INTEGER
DECLARE @data_length INT
DECLARE @ptrval BINARY(16)

-- read first blob value from a table
SELECT @data_length = DATALENGTH(image_col),
    @ptrval = TEXTPTR(image_col)
FROM images_table
WHERE image_id = 1

READTEXT images_table.image_col @ptrval 1 @data_length

-- pass that value to the AttachData procedure
EXEC @attach_id = master.dbmail.AttachData NULL, 'product.gif', @ptrval

-- read second blob value from a table
SELECT @data_length = DATALENGTH(image_col),
    @ptrval = TEXTPTR(image_col)
FROM images_table
WHERE image_id = 2

READTEXT images_table.image_col @ptrval 1 @data_length

-- pass that value to the AttachData procedure
EXEC master.dbmail.AttachData @attach_id, 'product2.gif', @ptrval

-- read third blob value from a table
SELECT @data_length = DATALENGTH(image_col),
    @ptrval = TEXTPTR(image_col)
FROM images_table
WHERE image_id = 3

READTEXT images_table.image_col @ptrval 1 @data_length

-- pass that value to the AttachData procedure
EXEC master.dbmail.AttachData @attach_id, 'product3.gif', @ptrval

-- now, mail it
EXEC master.dbmail.SendMail 'user@domain.com',
    'Test message with attachments',
    'This message has one attachment',
    NULL,
    NULL,
    NULL,
    @attach_id
```

Example 5 (many messages, 1 attachment):

```

-- First, let's create a user-defined scalar Transact-SQL function that
-- we can call from SELECT statements
CREATE FUNCTION mySendMail( @recipient VARCHAR(30),
                           @subject VARCHAR(255),
                           @message VARCHAR(8000),
                           @file_name VARCHAR(255) )

RETURNS INT
AS
BEGIN
    DECLARE @attach_id INTEGER
    DECLARE @ret INT
    EXEC @attach_id = master.dbo.AttachFile NULL, @file_name

    EXEC @ret = master.dbo.SendMail @recipient, @subject,
                                   @message, 'custservice@domain.com',
                                   0, @attach_id

    RETURN (@ret)
END
go

-- Now, we can call our own send mail function
SELECT mySendMail(customer_email,
                  'New great product in our store',
                  'We are happy to offer the product that you have asked for.' +
                  'For details please see the attached picture.',
                  'product.gif')
FROM customer
WHERE customer_email IS NOT NULL
      AND status = 'A';
go

```

AttachData

Use this method to attach data already stored inside the database in TEXT and IMAGE columns to email messages sent using SendMail procedure. To attach external files, use the AttachFile procedure. The definition of AttachData procedure is shown below:

Definition

```

PROCEDURE dbo.AttachData(
    @id          NUMBER = NULL,
    @file_name   VARCHAR(255),
    @data        IMAGE )

```

Argument	Max size; Value range	Description
ID		Attachment group ID. In order to generate unique Attachment IDs, pass NULL as a parameter when calling AttachData or AttachFile for the first time for every email message. The called procedure will return a unique ID that you must use in

		subsequent calls when attaching more files or data to the same message.
File_Name	255	Name of the attached file as it will appear in the email message. The name must conform to standard file naming conventions and should not include the file path. The file name must be unique within a single email message.
Data	2GB	BLOB value (IMAGE) containing the actual attachment data.

Return values: Returns unique attachment group ID or returns -1 if an error occurs.



Important Note: New attachment group ID is returned if NULL is passed for the **ID** argument, otherwise the value of the **ID** argument is returned.



Usage Tips:

- If you want to attach more than one BLOB, call the AttachData procedure as many times as many BLOB values you want to add.
- To attach external files, use the AttachFile procedure.
- To attach both external files and data, call both procedures as many times as there are attachments you want to add. Note that every attachment will appear as a file in the resulting email message. Names of files in the resulting message will match names of files that you have specified as arguments for the functions.



Important notes:

- DB Mail temporarily stores all attachments in the MAIL_ATTACH table. One message can have up to 255 attachments. Every message must have a unique Attachment ID referencing attachments that belong to that message. Within this group of records referenced by the Attachment ID, every individual attachment must have a unique File ID and File Name. Both Attachment ID and File ID are used as the surrogate key.
- DB Mail automatically deletes all attachments from the MAIL_ATTACH table after the message is sent or queued for sending.
- You must create separate attachments for every email message. To send an identical message to multiple recipients either send it to a user group using group name as recipient address or specify all recipients as a comma-separated list. The maximum length of the list is limited by the size of VARCHAR variable in your SQL Server version (normally 8000 characters). Only one copy of attachments is needed in both cases. In all other cases you must create separate messages with separate attachments for every recipient.

Examples

See examples for the [AttachFile](#) procedure that also includes examples for the AttachData procedure.

CreateMailFile

Use this method to write various flat files to be attached to email messages or used with DB Mail fax procedures. The definition of CreateMailFile procedure is shown below:

Definition

```
PROCEDURE dbmail.CreateMailFile(
    @file_name    VARCHAR(255),
    @text         VARCHAR(8000),
    @append      INT )
```

Argument	Max size; Value range	Description
File_Name	255	Name of the file to write to. Make sure to specify full file name including path. If you omit the path DB Mail will create the specified file in the current directory which normally SQL Server's BINN directory.
Text	8000	Text to write to the file specified by the File_Name argument.
Append	0..1	Value of the Append argument specifies whether to create a new file or append value of the Text argument to an existing file. If a value of 0 is passed for the Append argument and the file already exists, CreateMailFile overwrites the existing file. If a value of 1 is passed for the Append argument and the file does not exist, CreateMailFile fails and an application error is raised.

Return values: 0 or positive number is returned if the procedure successfully completed. A negative number is returned if a file cannot be written.



Usage Tips:

- Use the @@ERROR global variable to check for errors. @@ERROR is set to 0 if the procedure executed successfully. If an error occurs, a non-zero number is returned. @@ERROR returns the number of the error message until another Transact-SQL statement is executed. You can view the text associated with an @@ERROR error number in the SYSMESSAGES system table.
- Call CreateMailFile as many times for as many chunks of text you want to write to the file.
- When calling CreateMailFile for the first time to write the very first chunk of text, specify 0 for the Append parameter. In subsequent calls specify 1.
- If you use CreateMailFile to write a file attachment for the AttachFile procedure or to write a fax file for the SendFax or SendFaxEx procedure and you do not need that file after calling one of the Send methods, you can delete it using DeleteMailFile procedure.

Examples

Example 1 (create email attachment):

```
DECLARE @attach_id INTEGER
```

```
EXEC master.dbmail.CreateMailFile 'c:\temp\report.htm',
  '<html><title>Test Report</title><body>
  <h2>Test Report</h2><hr>
  <table><tr><th bgcolor=black><font color=white>Col 1</th>
  <th bgcolor=black><font color=white>Col 2</th></tr>
  <tr><td>Value A</td><td align=right>1</td></tr>
  <tr><td>Value B</td><td align=right>2</td></tr>
  </table></body></html>', 0

EXEC @attach_id = master.dbmai.AttachFile NULL, 'c:\temp\report.htm'

EXEC master.dbmail.DeleteMailFile 'c:\temp\report.htm'

EXEC master.dbmail.SendMail('me@mycompany.com',
  'Test message with attachments',
  'The test report is attached. Please ignore this message.',
  'myname@mycompany.com',
  'text/plain',
  1,
  @attach_id
```

Example 2 (create fax document):

```
DECLARE @attach_id INTEGER

EXEC master.dbmail.CreateMailFile 'c:\temp\report.htm',
  '<html><title>Test Report</title><body>
  <h2>Test Report</h2><hr>
  <table><tr><th bgcolor=black><font color=white>Col 1</th>
  <th bgcolor=black><font color=white>Col 2</th></tr>
  <tr><td>Value A</td><td align=right>1</td></tr>
  <tr><td>Value B</td><td align=right>2</td></tr>
  </table></body></html>', 0

EXEC master.dbmail.SendFax '+1 (123) 222-3344',
  'Test fax subject',
  'c:\temp\report.htm',
  TRUE,
  'Generic',
  'Test recipient'

EXEC master.dbmail.DeleteMailFile 'c:\temp\report.htm'
```

DeleteMailFile

Use this method to delete files created using the [CreateMailFile](#) procedure.

Definition

```
PROCEDURE dbmail.DeleteMailFile(
  @file_name          VARCHAR(255) )
```

Argument	Max size; Value range	Description
File_Name	255	Name of the file to delete. Make sure to specify full file name including path.

Return values: 0 or positive number is returned if procedure successfully completed. A negative number is returned if a file cannot be written.

Usage Tips:

Use the @@ERROR global variable to check for errors. @@ERROR is set to 0 if the procedure executed successfully. If an error occurs, a non-zero number is returned. @@ERROR returns the number of the error message until another Transact-SQL statement is executed. You can view the text associated with an @@ERROR error number using the SYSMESSAGES system table

Examples

See examples available in the CreateMailFile topic.

Sybase SQL Server, ASE, ASA


SendMail

Use this method to send email messages from your database. The definition of SendMail function is shown below:

Definition

```
PROCEDURE dbmail.SendMail(
    @recipients    VARCHAR(255),
    @subject       VARCHAR(255),
    @message       VARCHAR(255),
    @reply_to      VARCHAR(255),
    @content_type  VARCHAR(50),
    @priority      INT,
    @attachment_id INT,
    OUT @SQLMessage VARCHAR(255),
    OUT @result    INT)
```

Argument	Max Size; Value Range	Description
Recipients	255	Email address of message recipient. If you need to send the same message to multiple recipients, use comma to separate multiple recipient addresses. You can also specify valid email user groups. Email groups (also known as D-Lists or Distribution Lists) can be configured using your

		mail server administration tools.
Subject	255	Email message subject
Message	255	Email message text.  Tip: If you need to send a message whose text is longer than the maximum size of varchar data-type in your database you can use a special attachment file with the name <i>message</i> and no extension. DB Mail will use the contents of that file for the message text. The value of the Message parameter will be ignored in that case. For more information see Error! Not a valid result for table. topic.
Reply-to (optional)	255	Sender's email address. Note that this property is used with SMTP interface only. Your server must support relay forwarding if you want to set the sender's email address to a value different from the one specified in the DB Mail email configuration. When using MAPI and Lotus Notes interfaces, DB Mail ignores this property. Also, note that for both MAPI and Lotus Notes, your email messages will have sender's address specified in the profile of the user whose account you used to configure DB Mail Server mail options.
Content_Type (optional)	50	Message content type is one of text/plain , text/html , text/xml . Note that this property is used with SMTP interface only. When using MAPI and Lotus Notes interfaces, DB Mail ignores this property. If you specify NULL or omit this parameter, DB Mail uses the default value, which is text/plain .
Priority (optional)	0..2	Message processing priority takes a value of 0 , 1 , or 2 . Note that this property is used only when Message Queuing is enabled in DB Mail options. Messages having higher priority numbers are processed before messages having lower priority numbers. Email messages sent using SMTP protocol also inherit this priority attribute.
Attachment_ID (optional)		ID of the record or group of records in the MAIL_ATTACH table containing attachment data. Message attachment should be created using <i>AttachFile</i> and <i>AttachData</i> methods.
SQLMessage	255	Output variable used to return error description in case an error occurs
Result		Output variable to return procedure result

Return values:

Result returns unique message ID for the sent message (a positive number) if the procedure completes successfully.

If an error occurs, **Result** returns a negative number. Use **SQLMessage** argument to obtain the error description.

Examples

The following examples demonstrate how to send email messages using DB Mail functions.

Example 1:

The following EXECUTE statement will send email to all users who run batch jobs and whose database password is going to expire next Monday. This assumes that the BATCH_USER table contains the EXPIRE_DATE, the USERID and FNAME columns, which represent the expiry date of the password, the Mail email user id and the full user name.

```
DECLARE @Ret INT, @ErrorMessage VARCHAR(255)
EXEC sybsystemprocs.dbmail.SendMail 'batch_users@domain.com',
    'Password expiration',
    'Attention batch job owners, your database password will
     expire in 3 days. Be sure to update your batch jobs before
     that date. If you need assistance, reply to this message
     with your questions',
    'helpdesk@domain.com',
    'text/plain', 1, 0, @ErrorMessage, @Ret
```

Example 2:

This is a more advanced T-SQL example of a similar use of the dbmail.SendMail function:

```
/******
 * This procedure debits the specified account by specified amount
 * only if there are sufficient funds to cover the withdrawal, and if there
 * are not, it sends an email alert to the account manager.
 * This function returns new account balance.
 *****/
CREATE PROCEDURE sp_acct_debit (@acct_nbr CHAR(10), @debit_amt MONEY)
RETURNS INTEGER
AS
BEGIN
    DECLARE @acct_balance MONEY, @ret_code INTEGER
    SELECT @ret_code = -1

    SELECT @acct_balance = balance
    FROM accounts
    WHERE acct = @acct_nbr

    IF @acct_balance >= @debit_amt
    BEGIN
        SELECT @acct_balance = @acct_balance - @debit_amt
        UPDATE accounts
        SET balance = @acct_balance
        WHERE acct = @acct_nbr

        IF @@error = 0 SELECT @ret_code = 1 -- success
    END
    ELSE
        -- Insufficient funds.
        -- Send email notification to the account manager
        DECLARE @email VARCHAR(50), @message VARCHAR(200)
        DECLARE @Ret INT, @ErrorMessage VARCHAR(255)

        SELECT @email = m.email,
            @message = 'Time: ' + convert(varchar, GetDate()) + char(10) +
                'Account: ' + convert(varchar, @acct_nbr)
        FROM managers m, accounts a
        WHERE m.mgr_id = a.mgr_id
            AND a.acct = @acct_nbr
```



```

EXEC sybssystemprocs.dbmail.SendMail @email,
                                     'WARNING: Insufficient funds',
                                     @message, 'text/plain',
                                     'custservice@bank.com',
                                     1, 0, @ErrorMessage, @Ret

-- If email was sent successfully set ret_code to 0,
-- otherwise leave it as -1
IF @Ret = 1 SELECT @ret_code = 0 ELSE SELECT @retcode = -1
END

RETURN (@ret_code) -- return code 1 indicates success
                 -- 0 indicates insufficient funds and
                 --      successful notice
                 -- -1 indicates insufficient funds and failed
                 --      notice

END

```

Example 3 (sending email in HTML format):

The following EXECUTE statement will send emails to batch_users email group which includes users who run batch jobs. It uses HTML rather than plain text – note the use of the 'text/html' content type description:

```

DECLARE @Ret INT, @ErrorMessage VARCHAR(255)
EXEC sybssystemprocs.dbmail.SendMail 'batch_users@domain.com',
    'Password expiration',
    '<p>Attention batch job owners,</p>
    <p>Your database password will <b>expire</b> in 3 days.
    Be sure to update your batch jobs before that date.</p>
    <p><font color="#00255">If you need assistance,
    reply to this message with your questions</font></p>',
    'helpdesk@domain.com',
    'text/html', 1, 0, @ErrorMessage, @Err

```

AttachFile

Use this method to attach **virtual** files to email messages sent using the dbmail.SendMail procedure.



Important Note: Because Sybase does not currently support any methods for reading external files from database Transact-SQL and Java procedures, the DB Mail implementation for Sybase databases also does not support attaching external files directly.

As an alternative, you can write Sybase client programs to load external files into TEXT or IMAGE columns stored in database tables and then use the `AttachData` method to attach loaded files to messages. Such programs can be written in C, Java and a variety of other development systems. A number of examples of such programs are available on the Sybase web site and on the Internet.

Another alternative for writing and attaching text files is to use the `AttachFile` method together with the `CreateMailFile` method. The `CreateMailFile` method can be used to write virtual files stored in pieces in the MAIL_FILE database table. Then the `AttachFile` method can be used to attach such files to a email message sent using `SendMail` procedure.

The definition of `AttachFile` procedure is shown below:

Definition

```

PROCEDURE dbmail.AttachFile(
    @id                INT = NULL,
    @file_name         VARCHAR( 255 ),
    OUT @SQLMessage    VARCHAR( 255 ),
    OUT @result        INT )

```

Argument	Max size; Value range	Description
ID		Attachment group ID. In order to generate unique Attachment IDs, pass NULL as a parameter when calling ATTACH_DATA or ATTACH_FILE for the first time for every email message. The called procedure will return a unique ID that you must use in subsequent calls when attaching more files or data to the same message.
File_Name	255	Name of the attached file as it will appear in the email message. The name must conform to standard file naming conventions and should not include the file path. File name must be unique within single email message. File must be created using CreateMailFile procedure. File names are case –sensitive.
SQLMessage	255	Output variable used to return error description in case an error occurs
Result		Output variable to return procedure result

Return values:

Result returns unique attachment group ID (a positive number) if the procedure completes successfully.

If an error occurs, **Result** returns a negative number. Use **SQLMessage** argument to obtain the error description.



Important Note: The New attachment group ID is returned if NULL is passed for the **ID** argument, otherwise the value of the **ID** argument is returned.

**Usage Tips:**

- If you want to attach more than one file, call the AttachFile procedure as many times for as many files you want to add.
- To attach data already stored inside the database in TEXT and IMAGE columns, use the AttachData procedure.
- To attach both virtual files and data, call both procedures as many times as there are attachments you want to add. Note that every attachment will appear as a file in the resulting email message. Names of files in the resulting message will match names of files that you have specified as arguments for the procedures.

**Important notes:**

- DB Mail temporarily stores all attachments in the MAIL_ATTACH table. One message can have up to 255 attachments. Every message must have a unique Attachment ID referencing attachments that belong to that message. Within this group of records referenced by the Attachment ID, every individual attachment must have a unique File ID and File Name. Both Attachment ID and File ID are used as the surrogate key.
- DB Mail automatically deletes all attachments from the MAIL_ATTACH table after the message is sent or queued for sending.
- You must create separate attachments for every email message. To send an identical message to multiple recipients either send it to a user group using group name as recipient address or specify all recipients as a comma-separated list. The maximum length of the list is limited by the size of VARCHAR variable in your Sybase database version (normally 255 characters). Only one copy of attachments is needed in both cases. In all other cases you must create separate messages with separate attachments for every recipient.

Examples

The following examples demonstrate how to send email messages with attachments using DB Mail procedures. Here are several brief examples for attaching an external file and data from IMAGE columns.

Example 1 (1 message, 1 file attachment):

```
DECLARE @attach_id INTEGER
DECLARE @Ret INT, @ErrorMessage VARCHAR(255)

EXEC sybssystemprocs.dbmail.AttachFile NULL,
    'c:\images\product.gif',
    @ErrorMessage, @attach_id

EXEC sybssystemprocs.dbmail.SendMail 'user@domain.com',
    'Test message with attachments',
    'This message has one attachment',
    'myname@domain.com',
    'text/plain', 0, @attach_id, @ErrorMessage, @Err
```

Example 2 (1 message, 3 file attachments):

```
DECLARE @attach_id INTEGER
DECLARE @Ret INT, @ErrorMessage VARCHAR(255)

EXEC sybssystemprocs.dbmail.AttachFile NULL,
    'c:\images\product1.gif',
    @ErrorMessage, @attach_id
EXEC sybssystemprocs.dbmail.AttachFile @attach_id,
    'c:\images\product2.gif',
    @ErrorMessage, @Err
EXEC sybssystemprocs.dbmail.AttachFile @attach_id,
    'c:\images\products3.gif',
    @ErrorMessage, @Err

EXEC sybssystemprocs.dbmail.SendMail 'user@domain.com',
    'Test message with attachments',
    'This message has three attachments',
    'myname@domain.com',
```

```
'text/plain', 0, @attach_id, @ErrorMessage, @Err
```

Example 3 (1 message, 1 BLOB attachment):

```
DECLARE @attach_id INTEGER
DECLARE @ptrval BINARY(16)
DECLARE @Ret INT, @ErrorMessage VARCHAR(255)

-- read blob value from a table
SELECT @ptrval = TEXTPTR(image_col)
FROM images_table
WHERE image_id = 21

-- pass that value to the AttachData procedure
EXEC sybsystemprocs.dbmail.AttachData NULL,
    'product.gif', @ptrval,
    @ErrorMessage, @attach_id

-- now, mail it
EXEC sybsystemprocs.dbmail.SendMail 'user@domain.com',
    'Test message with attachments',
    'This message has one attachment',
    'myname@domain.com',
    'text/plain', 0, @attach_id, @ErrorMessage, @Err
```

Example 4 (1 message, 3 BLOB attachments):

```
DECLARE @attach_id INTEGER
DECLARE @ptrval BINARY(16)
DECLARE @Ret INT, @ErrorMessage VARCHAR(255)

-- read first blob value from a table
SELECT @ptrval = TEXTPTR(image_col)
FROM images_table
WHERE image_id = 1

-- pass that value to the AttachData procedure
EXEC sybsystemprocs.dbmail.AttachData NULL,
    'product.gif', @ErrorMessage, @attach_id

-- read second blob value from a table
SELECT @ptrval = TEXTPTR(image_col)
FROM images_table
WHERE image_id = 2

-- pass that value to the AttachData procedure
EXEC sybsystemprocs.dbmail.AttachData @attach_id,
    'product2.gif', @ErrorMessage, @Err

-- read third blob value from a table
SELECT @ptrval = TEXTPTR(image_col)
FROM images_table
WHERE image_id = 3

-- pass that value to the AttachData procedure
EXEC sybsystemprocs.dbmail.AttachData @attach_id,
    'product3.gif', @ErrorMessage, @Err

-- now, mail it
```

```
EXEC sybssystemprocs.dbmail.SendMail 'user@domain.com',
    'Test message with attachments',
    'This message has one attachment',
    'myname@domain.com',
    'text/plain', 0, @attach_id, @ErrorMessage, @Err
```

AttachData

Use this method to attach data already stored inside the database in TEXT and IMAGE columns to email messages sent using SendMail procedure. To attach virtual files, use the AttachFile procedure. The definition of AttachData procedure is shown below:

Definition


```
PROCEDURE dbmail.AttachData(
    @id                NUMBER,
    @file_name         VARCHAR( 255 ),
    @data_ptr          VARBINARY( 16 ),
    OUT @SQLMessage    VARCHAR( 255 ),
    OUT @result        INT)
```

Argument	Max size; Value range	Description
ID		Attachment group ID. In order to generate unique Attachment IDs, pass NULL as a parameter when calling AttachData or AttachFile for the first time for every email message. The called procedure will return a unique ID that you must use in subsequent calls when attaching more files or data to the same message.
File_Name	255	Name of the attached file as it will appear in the email message. The name must conform to standard file naming conventions and should not include the file path. File name must be unique within a single email message.
Data_ptr	16	Data pointer to BLOB value (TEXT or IMAGE) as returned by <i>TEXTPTR</i> system function.
SQLMessage	255	Output variable used to return error description in case an error occurs
Result		Output variable to return procedure result

Return values:

Result returns unique attachment group ID (a positive number) if the procedure completes successfully.

If an error occurs, **Result** returns a negative number. Use **SQLMessage** argument to obtain the error description.

 **Important note:** The New attachment group ID is returned if NULL is passed for the **ID** argument, otherwise the value of the **ID** argument is returned.

 **Usage Tips:**

- If you want to attach more than one BLOB, call the AttachData procedure as many times as many BLOB values you want to add.
- To attach external files, use the AttachFile procedure.
- To attach both external files and data, call both procedures as many times as there are attachments you want to add. Note that every attachment will appear as a file in the resulting email message. Names of files in the resulting message will match names of files that you have specified as arguments for the functions.

 **Important notes:**

- DB Mail temporarily stores all attachments in the MAIL_ATTACH table. One message can have up to 255 attachments. Every message must have a unique Attachment ID referencing attachments that belong to that message. Within this group of records referenced by the Attachment ID, every individual attachment must have a unique File ID and File Name. Both Attachment ID and File ID are used as the surrogate key.
- DB Mail automatically deletes all attachments from the MAIL_ATTACH table after the message is sent or queued for sending.
- You must create separate attachments for every email message. To send an identical message to multiple recipients either send it to a user group using group name as recipient address or specify all recipients as a comma-separated list. The maximum length of the list is limited by the size of VARCHAR variable in your Sybase database version (normally 255 characters). Only one copy of attachments is needed in both cases. In all other cases you must create separate messages with separate attachments for every recipient.

Examples

See examples for the [AttachFile](#) procedure that also includes examples for the AttachData procedure.

CreateMailFile

Use this method to create virtual text (TXT, HTML, XML, etc...) files stored internally in MAIL_FILE table. Such virtual files can be attached to email messages and used with DB Mail fax procedures just as if they were regular external files. The definition of CreateMailFile procedure is shown below:

Definition

```
PROCEDURE dbmail.CreateMailFile(
    @file_name          VARCHAR( 255 ),
    @text               VARCHAR( 255 ),
    @append             INT,
    OUT @SQLMessage    VARCHAR( 255 ),
    OUT @result         INT)
```

Argument	Max size; Value range	Description
File_Name	255	Name of the file to write to. Do not specify file path!
Text	255	Text to write to the file specified by the File_Name argument.
Append	0..1	Value of the Append argument specifies whether to create a new file or append value of the Text argument to an existing file. If a value of 0 is passed for the Append argument and the file already exists, CreateMailFile overwrites the existing file. If a value of 1 is passed for the Append argument and the file does not exist, CreateMailFile fails and application error is raised.
SQLMessage	255	Output variable used to return error description in case an error occurs
Result		Output variable to return procedure result

Return values:

Result returns 1 if the procedure completes successfully.

If an error occurs, **Result** returns a negative number. Use **SQLMessage** argument to obtain the error description.

**Usage Tips:**

- Call CreateMailFile as many times as many chunks of text you want to write to the file. Because Sybase' implementation of VARCHAR data-type is limited to 255 characters, the size of a single chunk is also limited to 255 characters.
- When calling CreateMailFile for the first time to write the very first chunk of text, specify 0 for the Append parameter. In subsequent calls specify 1.
- If you use CreateMailFile to write a file attachment for the AttachFile procedure or to write a fax file for the SendFax or SendFaxEx procedure and you do not need that file after calling one of the Send methods, you can delete it using DeleteMailFile procedure.

Examples**Example 1 (create email attachment):**

```

DECLARE @Ret INT, @ErrorMessage VARCHAR(255)
DECLARE @attach_id INTEGER
DECLARE @SampleReport VARCHAR(255)

-- create new file and write first chunk
SELECT @SampleReport =
    '<html><title>Test Report</title><body>' + char(10) +
    '<h2>Test Report</h2><hr>' + char(10) +
    '<table><tr><th bgcolor=black><font color=white>Col 1</th>' +
    '<th bgcolor=black><font color=white>Col 2</th></tr>' + char(10)
EXEC sybssystemprocs.dbmail.CreateMailFile

```

```

'report.htm', @SampleReport, 0, @ErrorMessage, @Ret

-- append second chunk to the same file
SELECT @SampleReport =
    '<tr><td>Value A</td><td align=right>1</td></tr>' + char(10) +
    '<tr><td>Value B</td><td align=right>2</td></tr>' + char(10) +
    '</table></body></html>'
EXEC sybsystemprocs.dbmail.CreateMailFile
    'report.htm', @SampleReport, 1, @ErrorMessage, @Ret

-- now, let's attach the entire file
EXEC @attach_id = sybsystemprocs.dbmai.AttachFile NULL,
    'report.htm', @ErrorMessage, @Ret

-- we do not need this file anymore, let's delete it
EXEC sybsystemprocs.dbmail.DeleteMailFile
    'report.htm', @ErrorMessage, @Ret

-- send mew email with the attached file
EXEC sybsystemprocs.dbmail.SendMail('me@mycompany.com',
    'Test message with attachments',
    'The test report is attached. Please ignore this message.',
    'myname@mycompany.com',
    'text/plain', 1, @attach_id, @ErrorMessage, @Ret

```

Example 2 (create fax document):

```

DECLARE @Ret INT, @ErrorMessage VARCHAR(255)
DECLARE @attach_id INTEGER
DECLARE @SampleReport VARCHAR(255)

-- create new file and write first chunk
SELECT @SampleReport =
    '<html><title>Test Report</title><body>' + char(10) +
    '<h2>Test Report</h2><hr>' + char(10) +
    '<table><tr><th bgcolor=black><font color=white>Col 1</th>' +
    '<th bgcolor=black><font color=white>Col 2</th></tr>' + char(10)
EXEC sybsystemprocs.dbmail.CreateMailFile
    'report.htm', @SampleReport, 0, @ErrorMessage, @Ret

-- append second chunk to the same file
SELECT @SampleReport =
    '<tr><td>Value A</td><td align=right>1</td></tr>' + char(10) +
    '<tr><td>Value B</td><td align=right>2</td></tr>' + char(10) +
    '</table></body></html>'
EXEC sybsystemprocs.dbmail.CreateMailFile
    'report.htm', @SampleReport, 1, @ErrorMessage, @Ret

-- now, let's send the fax
EXEC sybsystemprocs.dbmail.SendFax '+1 (123) 222-3344',
    'Test fax subject',
    'report.htm',
    TRUE,
    'Generic',
    'Test recipient',
    @ErrorMessage, @Ret

-- we do not need this file anymore, let's delete it
EXEC sybsystemprocs.dbmail.DeleteMailFile
    'report.htm', @ErrorMessage, @Ret

```


DeleteMailFile

Use this method to delete virtual files created using [CreateMailFile](#) procedure.

Definition

```
PROCEDURE dbmail.DeleteMailFile(
    @file_name          VARCHAR( 255 ),
    OUT @SQLMessage     VARCHAR( 255 ),
    OUT @result         INT)
```

Argument	Max size; Value range	Description
File_Name	255	Name of the file to delete. Make sure to specify file name exactly as it was specified with the CreateMailFile procedure. File names are case-sensitive.
SQLMessage	255	Output variable used to return error description in case an error occurs
Result		Output variable to return procedure result

Return values:

Result returns 1 if the procedure completes successfully.

If an error occurs, **Result** returns a negative number. Use **SQLMessage** argument to obtain the error description.

Examples

See examples available in the [CreateMailFile](#) topic.

IBM DB2

SendMail

Use this method to send email messages from your database. The definition of SendMail function is shown below:


Definition

```
PROCEDURE dbmail.SendMail(
    recipients          VARCHAR( 32672 ),
    subject             VARCHAR( 255 ),
    message             VARCHAR( 32672 ),
```

```

reply_to          VARCHAR( 255 ),
content_type     VARCHAR( 50 ),
priority         INT,
attachment_id    INT,
OUT SQLMessage   VARCHAR( 1000 ),
OUT result       INT)

```

Argument	Max Size; Value Range	Description
Recipients	32672	Email address of message recipient. If you need to send the same message to multiple recipients, use comma to separate multiple recipient addresses. You can also specify valid email user groups. Email groups (also known as D-Lists or Distribution Lists) can be configured using your mail server administration tools.
Subject	255	Email message subject
Message	32672	Email message text.  Tip: If you need to send a message whose text is longer than the maximum size of varchar data-type in your database you can use a special attachment file with the name <i>message</i> and no extension. DB Mail will use the contents of that file for the message text. The value of the Message parameter will be ignored in that case. For more information see Error! Not a valid result for table. topic.
Reply-to (optional)	255	Sender's email address. Note that this property is used with SMTP interface only. Your server must support relay forwarding if you want to set the sender's email address to a value different from the one specified in the DB Mail email configuration. When using MAPI and Lotus Notes interfaces, DB Mail ignores this property. Also, note that for both MAPI and Lotus Notes, your email messages will have sender's address specified in the profile of the user whose account you used to configure DB Mail Server mail options.
Content_Type (optional)	50	Message content type is one of text/plain , text/html , text/xml . Note that this property is used with SMTP interface only. When using MAPI and Lotus Notes interfaces, DB Mail ignores this property. If you specify NULL or omit this parameter, DB Mail uses the default value, which is text/plain .
Priority (optional)	0..2	Message processing priority takes on a value of 0 , 1 , or 2 . Note that this property is used only when Message Queuing is enabled in DB Mail options. Messages having higher number have higher priority and processed before messages having lower priority. Email messages sent using SMTP protocol also inherit this priority attribute.
Attachment_ID (optional)		ID of the record or group of records in the MAIL_ATTACH table containing attachment data. Message attachment should be created using <i>AttachFile</i> and <i>AttachData</i> methods.
SQLMessage	1000	Output variable used to return error description in case an error occurs
Result		Output variable to return procedure result

Return values:

Result returns unique message ID for the sent message (a positive number) if the procedure completes successfully.

If an error occurs, **Result** returns a negative number. Use **SQLMessage** argument to obtain the error description.

Examples

The following examples demonstrate how to send email messages using DB Mail functions.

Example 1:

The following CALL statement will send email to all users who run batch jobs and whose database password is going to expire next Monday. This assumes that the BATCH_USER table contains the EXPIRE_DATE, the USERID and FNAME columns, which represent the expiry date of the password, the Mail email user id and the full user name.

```
DECLARE Ret INT;
DECLARE ErrMessage VARCHAR(1000);

CALL dbmail.SendMail( 'batch_users@domain.com',
    'Password expiration',
    'Attention batch job owners, your database password will
     expire in 3 days. Be sure to update your batch jobs before
     that date. If you need assistance, reply to this message
     with your questions',
    'helpdesk@domain.com',
    'text/plain', 1, 0, ErrMessage, Ret);
```

Example 2:

This is a more advanced example of a similar use of the dbmail.SendMail procedure:

```
/******
 * This procedure debits the specified account by specified amount
 * only if there are sufficient funds to cover the withdrawal, and if there
 * are not, it sends an email alert to the account manager.
 * This function returns new account balance.
 *****/
CREATE PROCEDURE sp_acct_debit (acct_nbr CHAR(10), debit_amt DECIMAL(11,2))
LANGUAGE SQL
BEGIN
    DECLARE acct_balance DECIMAL(11,2);
    DECLARE ret_code INTEGER;
    DECLARE email VARCHAR(50);
    DECLARE message VARCHAR(200);
    DECLARE errors VARCHAR(1000);

    SET ret_code = -1;

    SELECT balance
    INTO acct_balance
    FROM accounts
    WHERE acct = acct_nbr;

    IF acct_balance >= debit_amt THEN
```

```

SET acct_balance = acct_balance - debit_amt;

UPDATE accounts
SET balance = acct_balance
WHERE acct = acct_nbr;

IF @@error = 0 THEN SET @ret_code = 1; END IF; -- success
ELSE
-- Insufficient funds.
-- Send email notification to the account manager

SELECT m.email,
       'Time: ' || char(CURRENT DATE) || chr(10) ||
       'Account: ' || char(acct_nbr)
INTO email, message
FROM managers m, accounts a
WHERE m.mgr_id = a.mgr_id
      AND a.acct = acct_nbr;

CALL dbmail.SendMail( email,
                      'WARNING: Insufficient funds',
                      message, 'text/plain',
                      'custservice@bank.com',
                      1, 0, errors, ret_code );

-- If email was sent successfully set ret_code to 0,
-- otherwise leave it as -1
IF ret_code = 1 THEN
    SET ret_code = 0;
ELSE
    SET retcode = -1;
END IF;
END;

RETURN ret_code; -- return code 1 indicates success
                -- 0 indicates insufficient funds and
                --    successful notice
                -- -1 indicates insufficient funds and failed
                --    notice
END

```

Example 3 (sending email in HTML format):

The following EXECUTE statement will send emails to batch_users email group which includes users who run batch jobs. It uses HTML rather than plain text – note the use of the 'text/html' content type description:

```

DECLARE @Ret INT, @ErrMsg VARCHAR(255)
EXEC dbmail.SendMail 'batch_users@domain.com',
    'Password expiration',
    '<p>Attention batch job owners,</p>
    <p>Your database password will <b>expire</b> in 3 days.
    Be sure to update your batch jobs before that date.</p>
    <p><font color="#00255">If you need assistance,
    reply to this message with your questions</font></p>',
    'helpdesk@domain.com',
    'text/html', @ErrMsg, @Err

```

AttachFile

Use this method to attach external files to email messages sent using the `dbmail.SendMail` procedure.

The definition of `AttachFile` procedure is shown below:

Definition

```
PROCEDURE dbmail.AttachFile(
    id                INT,
    file_name         VARCHAR(255),
    OUT SQLMessage    VARCHAR(1000),
    OUT result        INT)
```

Argument	Max size; Value range	Description
ID		Attachment group ID. In order to generate unique Attachment IDs, pass NULL as a parameter when calling the <code>AttachData</code> or <code>AttacheFile</code> for the first time for every email message. The called procedure will return a unique ID that you must use in subsequent calls when attaching more files or data to the same message.
File_Name	255	Name of the attached file as it will appear in the email message. Name must conform to standard file naming conventions and should not include file path. File name must be unique within single email message.
SQLMessage	1000	Output variable used to return error description in case an error occurs
Result		Output variable to return procedure result

Return values:

Result returns unique attachment group ID (a positive number) if the procedure completes successfully.

If an error occurs, **Result** returns a negative number. Use **SQLMessage** argument to obtain the error description.



Important note: The New attachment group ID is returned if NULL is passed for the **ID** argument, otherwise the value of the **ID** argument is returned.



Usage Tips:

- If you want to attach more than one file, call the `AttachFile` procedure as many times for as many files you want to add.
- To attach data already stored inside the database in BLOB columns, use the `AttachData` procedure.
- To attach both virtual files and data, call both procedures as many times as there are attachments you want to add. Note that every attachment will appear as a file in the resulting email message. Names of files in the resulting message will match names of files that you have specified as arguments for the procedures.

**Important notes:**

- DB Mail temporarily stores all attachments in the MAIL_ATTACH table. One message can have up to 255 attachments. Every message must have a unique Attachment ID referencing attachments that belong to that message. Within this group of records referenced by the Attachment ID, every individual attachment must have a unique File ID and File Name. Both Attachment ID and File ID are used as the surrogate key.
- DB Mail automatically deletes all attachments from the MAIL_ATTACH table after the message is sent or queued for sending.
- You must create separate attachments for every email message. To send an identical message to multiple recipients either send it to a user group using group name as recipient address or specify all recipients as a comma-separated list. The maximum length of the list is limited by the size of VARCHAR variable in your DB2 version (normally 32765 characters). Only one copy of attachments is needed in both cases. In all other cases you must create separate messages with separate attachments for every recipient.

Examples

The following examples demonstrate how to send email messages with attachments using DB Mail functions. Here are brief examples for attaching an external file and data from BLOB columns.

Example 1 (DB2 SQL, 1 message, 1 file attachment):

```
BEGIN
  DECLARE attach_id INTEGER;
  DECLARE errors VARCHAR(1000);

  CALL dbmail.AttachFile(NULL, '/apps/images/product.gif',
                        errors, attach_id);

  CALL dbmail.SendMail('user@domain.com',
                      'Test message with attachments',
                      'This message has one attachment',
                      NULL, NULL, NULL, attach_id);
END
```

Example 2 (DB2 SQL, 1 message, 3 file attachments):

```
BEGIN
  DECLARE attach_id INTEGER;
  DECLARE errors VARCHAR(1000);

  CALL dbmail.AttachFile(NULL, '/apps/images/product1.gif',
                        errors, attach_id);

  CALL dbmail.AttachFile(attach_id, '/apps/images/product2.gif',
                        errors, attach_id);

  CALL dbmail.AttachFile(attach_id, '/apps/images/product3.gif',
                        errors, attach_id);

  CALL dbmail.SendMail('user@domain.com',
                      'Test message with attachments',
                      'This message has three attachments',
```

```

        NULL, NULL, NULL, attach_id);
END

```

Example 3 (DB2 SQL, 1 message, 1 BLOB attachment):

```

BEGIN
  DECLARE attach_id INTEGER;
  DECLARE errors VARCHAR(1000);
  DECLARE data BLOB(1M);

  SELECT image INTO data
  FROM scan_documents
  WHERE doc_type = 'INVOICE' AND doc_key = 1;

  CALL dbmail.AttachData(NULL, 'invoice1.gif', data,
                        errors, attach_id);

  CALL dbmail.SendMail('user@domain.com',
    'Test message with attachments',
    'This message has one attachment',
    NULL, NULL, NULL, attach_id);
END

```

Example 4 (DB2 SQL, 1 message, 3 BLOB attachments):

```

BEGIN
  DECLARE attach_id INTEGER;
  DECLARE ret_code INTEGER;
  DECLARE errors VARCHAR(1000);
  DECLARE data BLOB(1M);

  SELECT image INTO data
  FROM scan_documents
  WHERE doc_type = 'INVOICE' AND doc_key = 1;

  CALL dbmail.AttachData(NULL, 'invoice1.gif', data,
                        errors, attach_id);

  SELECT image INTO data
  FROM scan_documents
  WHERE doc_type = 'INVOICE' AND doc_key = 2;

  CALL dbmail.AttachData(attach_id, 'invoice2.gif', data,
                        errors, attach_id);

  SELECT image INTO data
  FROM scan_documents
  WHERE doc_type = 'INVOICE' AND doc_key = 3;

  CALL dbmail.AttachData(attach_id, 'invoice3.gif', data,
                        errors, attach_id);

  CALL dbmail.SendMail('user@domain.com',
    'Test message with attachments',
    'This message has 3 attachments',
    NULL, NULL, NULL, attach_id, errors, ret_code);
END

```

Example 5 (DB2 SQL, many messages, 1 attachment):

```

BEGIN
  DECLARE attach_id INTEGER;
  DECLARE ret_code INTEGER;
  DECLARE errors VARCHAR(1000);
  DECLARE cust_email VARCHAR(100);
  DECLARE cust_name VARCHAR(100);

  DECLARE c1 CURSOR FOR
    SELECT customer_name, customer_email
    FROM customer
    WHERE customer_email IS NOT NULL
    AND status = 'A';
  /* Open cursor */
  OPEN c1;

  /* loop through all customer records */
  fetch_loop: LOOP

    /* fetch customer email and name
    FETCH c1 INTO cust_name, cust_email;
    IF (SQLCODE != 0) THEN LEAVE fetch_loop; END IF;

    /* create attachment */
    CALL dbmail.AttachFile(NULL, 'product.bmp', errors, attach_id) ;

    /* send message */
    CALL dbmail.SendMail(cust_email,
      'New great product in our store',
      'Dear ' || cust_name || ', ' || chr(10) || chr(10) ||
      'We are happy to offer the product that you have asked for.' ||
      'For details please see the attached picture.'
      'sales@ourcompany.com',
      'text/plain', 0, attach_id, errors, ret_code);
  LOOP;

  CLOSE c1;
END

```

AttachData

Use this method to attach data already stored inside the database in BLOB columns to email messages sent using SendMail procedure. To attach virtual files, use the AttachFile procedure. The definition of AttachData procedure is shown below:

Definition

```

PROCEDURE dbmail.AttachData(
  id                NUMBER,
  file_name         VARCHAR(255),
  data              BLOB(10M),
  UT SQLMessage    VARCHAR(1000),
  OUT result        INT)

```


Argument	Max size; Value range	Description
ID		Attachment group ID. In order to generate unique Attachment IDs, pass NULL as a parameter when calling AttachData or AttachFile for the first time for every email message. The called function will return a unique ID that you must use in subsequent calls when attaching more files or data to the same message.
File_Name	255	Name of the attached file as it will appear in the email message. The name must conform to standard file naming conventions and should not include the file path. File name must be unique within single email message.
Data	10M	Data pointer to BLOB value.
SQLMessage	1000	Output variable used to return error description in case an error occurs
Result		Output variable to return procedure result

Return values:

Result returns unique attachment group ID (a positive number) if the procedure completes successfully.

If an error occurs, **Result** returns a negative number. Use **SQLMessage** argument to obtain the error description.



Important Note: New attachment group ID is returned if NULL is passed for the **ID** argument, otherwise the value of the **ID** argument is returned.

**Usage Tips:**

- If you want to attach more than one blob, call the AttachData procedure as many times as many BLOB values you want to add.
- To attach external files, use the AttachFile procedure.
- To attach both external files and data, call both procedures as many times as there are attachments you want to add. Note that every attachment will appear as a file in the resulting email message. Names of files in the resulting message will match names of files that you have specified as arguments for the functions.

**Important notes:**

- DB Mail temporarily stores all attachments in the MAIL_ATTACH table. One message can have up to 255 attachments. Every message must have a unique Attachment ID referencing attachments that belong to that message. Within this group of records referenced by the Attachment ID every individual attachment must have a unique File ID and File Name. Both Attachment ID and File ID are used as the surrogate key.

- DB Mail automatically deletes all attachments after the message is sent or queued for sending. You must create separate attachments for every email message to be sent. If you intend to send an identical message to multiple recipients you should either send such message to a user group or specify all recipients as a comma-separated list. The maximum length of the list is limited by the size of VARCHAR variable in your DB2 version (normally 32762 characters). Otherwise, you must create separate messages with separate attachments for every recipient.

Examples

See examples for the `AttachFile` procedure that also includes examples for the `AttachData` procedure.

CreateMailFile

Use this method to create flat files such as TXT, HTML, XML, etc... on the database server computer.. Created files can then be attached to email messages and used with DB Mail fax procedures. The definition of `CreateMailFile` is shown below:

Definition

```
PROCEDURE dbmail.CreateMailFile(
    file_name      VARCHAR( 255 ),
    text           VARCHAR( 32672 ),
    append         INT,
    OUT SQLMessage VARCHAR( 1000 ),
    OUT result     INT)
```

Argument	Max size; Value range	Description
File_Name	255	Name of the file to write to. Do not specify file path!
Text	32672	Text to write to the file specified by the File_Name argument.
Append	0..1	Value of the Append argument specifies whether to create a new file or append value of the Text argument to an existing file. If a value of 0 is passed for the Append argument and the file already exists, <code>CreateMailFile</code> overwrites the existing file. If a value of 1 is passed for the Append argument and the file does not exist, <code>CreateMailFile</code> fails an application error is raised.
SQLMessage	1000	Output variable used to return error description in case an error occurs
Result		Output variable to return procedure result

Return values:

Result returns 1 if the procedure completes successfully.

If an error occurs, **Result** returns a negative number. Use **SQLMessage** argument to obtain the error description.

 **Usage Tips:**

- Call CreateMailFile as many times as many chunks of text you want to write to the file.
- When calling CreateMailFile for the first time to write the very first chunk of text, specify 0 for the Append parameter. In subsequent calls specify 1.
- If you use CreateMailFile to write a file attachment for the AttachFile procedure or to write a fax file for the SendFax or SendFaxEx procedure and you do not need that file after calling one of the Send methods you can delete it using DeleteMailFile procedure.

Examples**Example 1 (DB2 SQL; create email attachment):**

```
BEGIN
  DECLARE attach_id INTEGER;
  DECLARE ret_code INTEGER;
  DECLARE errors VARCHAR(1000);
  DECLARE report VARCHAR(4000);

  /* create dynamic HTML report */
  SET report = '<html><title>Test Report</title><body>' || chr(10) ||
    '<h2>Test Report</h2><hr>' || chr(10) ||
    '<table><tr>' || chr(10) ||
    '<th bgcolor=black><font color=white>Col 1</th>' || chr(10) ||
    '<th bgcolor=black><font color=white>Col 2</th></tr>' || chr(10) ||
    '<tr><td>Value A</td><td align=right>1</td></tr>' || chr(10) ||
    '<tr><td>Value B</td><td align=right>2</td></tr>' || chr(10) ||
    '</table></body></html>';

  CALL dbmail.CreateMailFile( '/maildir/report.htm', report,
    0, errors, ret_code );

  /* create email attachment from report file */
  CALL dbmai.AttachFile( NULL, '/maildir/report.htm', errors, attach_id );

  /* delete temporary report file */
  CALL dbmail.DeleteMailFile( '/maildir/report.htm', errors, ret_code );

  /* send report by email */
  CALL dbmail.SendMail('me@mycompany.com',
    'Test message with attachments',
    'The test report is attached. Please ignore this message.',
    'myname@mycompany.com', 'text/plain', 1, attach_id,
    errors, ret_code );
END
```

Example 2 (DB2 SQL; create fax document):

```
BEGIN
  DECLARE ret_code INTEGER;
  DECLARE errors VARCHAR(1000);
  DECLARE report VARCHAR(4000);

  /* create dynamic HTML report */
  SET report = '<html><title>Test Report</title><body>' || chr(10) ||
    '<h2>Test Report</h2><hr>' || chr(10) ||
```

```

'<table><tr>' || chr(10) ||
'<th bgcolor=black><font color=white>Col 1</th>' || chr(10) ||
'<th bgcolor=black><font color=white>Col 2</th></tr>' || chr(10) ||
'<tr><td>Value A</td><td align=right>1</td></tr>' || chr(10) ||
'<tr><td>Value B</td><td align=right>2</td></tr>' || chr(10) ||
'</table></body></html>';

CALL dbmail.CreateMailFile( '/maildir/report.htm', report,
                           0, errors, ret_code );

/* send report by fax */
CALL dbmail.SendFax('+1 (123) 222-3344',
                   'Test fax subject', '/maildir/report.htm'
                   'Generic', 'Test recipient',
                   errors, ret_code );

/* delete temporary report file */
CALL dbmail.DeleteMailFile( '/maildir/report.htm', errors, ret_code );

END

```

DeleteMailFile

Use this method to delete files created using `CreateMailFile` procedure.

Definition

```

PROCEDURE dbmail.DeleteMailFile(
    file_name          VARCHAR(255),
    OUT SQLMessage     VARCHAR(1000),
    OUT result         INT)

```

Argument	Max size; Value range	Description
File_Name	255	Name of the file to delete. Make sure to specify file name exactly as it was specified with the <code>CreateMailFile</code> procedure.
SQLMessage	1000	Output variable used to return error description in case an error occurs
Result		Output variable to return procedure result

Return values:

Result returns 1 if the procedure completes successfully.

If an error occurs, **Result** returns a negative number. Use **SQLMessage** argument to obtain the error description

Examples

See examples available in the [CreateMailFile](#) topic.

CHAPTER 7, Sending SMS/pager messages

Overview

To send alphanumeric pager and cell phone messages from database applications use the *dbmail.SendPage* method. This method can be used to send page messages to single or multiple message recipients.

For detailed method descriptions and usage examples specific to your database system, refer to the following topics in this chapter.

Oracle

SEND_PAGE

Use this method to send alphanumeric pager and cell phone messages from your database. The definition of SEND_PAGE function is shown below:

Definition

```
db_mail.send_page(
    recipients          VARCHAR2,
    message             VARCHAR2 )
RETURN NUMBER
```

Argument	Max Size; Value Range	Description
Recipients	4000	Pager or cell phone number of the message recipient including area code. If you need to send the same message to multiple recipients, use comma to separate multiple recipient numbers.
Message	255	Message text.

Return values:

0 - Success.

1 - Timed out. This procedure can timeout either because it cannot get a lock on the pipe, or because the pipe remains too full to be used.

3 - An interrupt occurred.

 **Usage Tips:**

If you add a call to the SEND_PAGE function from a multi-row SQL statement such as SELECT ... FROM ... TABLE, Oracle will invoke the SEND_PAGE function as many times as many rows are affected by the statement.

The following example demonstrates how to send pager messages using a DB Mail function.

Example:

The following SELECT statement will send cell phone messages to all system administrators.

```
SELECT db_mail.send_page(phone_number,
    'Free space in the database is critically low. ' ||
    'Your immediate attention is required' )
FROM sysadmins;
```

Microsoft SQL Server

SendPage

Use this method to send alphanumeric pager and cell phone messages from your database. The definition of SendPage procedure is shown below:

Definition

```
PROCEDURE dbmail.SendPage(
    @recipients    VARCHAR(8000),
    @message       VARCHAR(255))
```

Argument	Max Size; Value Range	Description
Recipients	8000	Pager or cell phone number of the message recipient including area code. If you need to send the same message to multiple recipients, use comma to separate multiple recipient numbers.
Message	255	Message text.

Return values: Returns unique message ID or returns -1 if an error occurs.

 **Usage Tips:**

Use the @@ERROR global variable to check for errors. @@ERROR is set to 0 if the procedure executed successfully. If an error occurs, a non-zero number is returned. @@ERROR returns the number of the error message until another Transact-SQL statement is executed. You can view the text associated with an @@ERROR error number in the SYSMESSAGES system table.

The following examples demonstrate how to send pager messages using DB Mail functions.

Example 1, single message

The following EXECUTE statement will send message messages to (111) 222-3344 pager number.

```
EXEC master.dbmail.SendPage '1112223344',
    'Free space in the database is critically low. Your immediate
attention is required'
```

Example 2, multiple messages sent in a loop

In this example we will send cell phone messages to all system administrators listed in the administrators table.

```
DECLARE @cell_phone VARCHAR(20)
DECLARE c1 CURSOR
FOR SELECT cell_phone
FROM administrators

OPEN c1
-- get first cell phone number
FETCH NEXT FROM c1 INTO @cell_phone

WHILE (@@fetch_status = 0)
BEGIN
    -- replace non-digits
    SET @sell_phone = replace(
        replace(
            replace(@cell_phone, '(', ''),
            ')', ''),
        ' ', '')

    -- send page
    EXEC master.dbmail.SendPage @sell_phone,
        'Free space in the database is critically low. Your immediate
attention is required'

    -- get next number
    FETCH NEXT FROM c1 INTO @cell_phone
END

CLOSE c1
DEALLOCATE c1
```

Sybase SQL Server, ASE, ASA

SendPage

Use this method to send alphanumeric pager and cell phone messages from your database. The definition of SendPage is shown below:

Definition

```
PROCEDURE dbmail.SendPage(
    @recipients    VARCHAR(255),
    @message       VARCHAR(255),
    OUT @SQLMessage VARCHAR(255),
```



```
OUT @result INT)
```

Argument	Max Size; Value Range	Description
Recipients	255	Pager or cell phone number of the message recipient including area code. If you need to send the same message to multiple recipients, use comma to separate multiple recipient numbers.
Message	255	Message text.
SQLMessage	255	Output variable used to return error description in case an error occurs
Result		Output variable to return procedure result

Return values:

Result returns unique message ID for the sent message (a positive number) if the procedure completes successfully.

If an error occurs, **Result** returns a negative number. Use **SQLMessage** argument to obtain the error description.

The following examples demonstrate how to send pager messages using DB Mail functions.

Example 1, single message

The following EXECUTE statement will send message to (111) 222-3344 pager number.

```
DECLARE @Ret INT, @ErrorMessage VARCHAR(255)
EXEC sybsystemprocs.dbmail.SendPage '2122134455',
    'Free space in the database is critically low. Your immediate
attention is required', @ErrorMessage, @Ret
```

Example 2, multiple messages sent in a loop (DB2 C example)

In this example we will send cell phone messages to all system administrators listed in the administrators table.

```
DECLARE @Ret INT, @ErrorMessage VARCHAR(255)
DECLARE @cell_phone VARCHAR(20)
DECLARE c1 CURSOR
FOR SELECT cell_phone
FROM administrators

OPEN c1
-- get first cell phone number
FETCH c1 INTO @cell_phone

WHILE (@@sqlstatus = 0)
BEGIN
    -- replace non-digits
    -- (note: str_replace function is available in 12.5.0.3 and later)
    SELECT @sell_phone = str_replace(
        str_replace(
            str_replace(@cell_phone, '(', ''),
            ')', ''),
```

```

        ' ', '' )
-- send page
EXEC sybssystemprocs.dbmail.SendPage @sell_phone,
    'Free space in the database is critically low. Your immediate
    attention is required', @ErrorMessage, @Ret

-- get next number
FETCH c1 INTO @cell_phone
END

CLOSE c1

```

IBM DB2

SendPage

Use this method to send alphanumeric pager and cell phone messages from your database. The definition of SendPage is shown below:

Definition

```

PROCEDURE dbmail.SendPage(
    recipients    VARCHAR( 32672 ),
    message       VARCHAR( 255 ),
    OUT SQLMessage VARCHAR( 1000 ),
    OUT result    INT )

```

Argument	Max Size; Value Range	Description
Recipients	32672	Pager or cell phone number of the message recipient including area code. If you need to send the same message to multiple recipients, use comma to separate multiple recipient numbers.
Message	255	Message text.
SQLMessage	1000	Output variable used to return error description in case an error occurs
Result		Output variable to return procedure result

Return values:

Result returns unique message ID for the sent message (a positive number) if the procedure completes successfully.

If an error occurs, **Result** returns a negative number. Use **SQLMessage** argument to obtain the error description.

Examples:

The following examples demonstrate how to send pager messages using DB Mail functions.

Example 1, single message (DB2 SQL)

The following CALL statement will send message to (111) 222-4455 pager number.

```
DECLARE Ret INT;
DECLARE ErrMessage VARCHAR(1000);

CALL dbmail.SendPage( '1112224455',
    'Free space in the database is critically low. Your immediate
    attention is required', ErrMessage, Ret);
```

Example 2, multiple messages sent in a loop (DB2 C)

This example DB2 external stored procedure (written in C language) will send cell phone messages to all system administrators listed in the staff table.

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

EXEC SQL INCLUDE SQLCA;

int main() {

    EXEC SQL BEGIN DECLARE SECTION;
        char phone[12] = {'\0'};
        char errors[1000] = {'\0'};
        int ret = 0;
    EXEC SQL END DECLARE SECTION;

    printf( "Sample C program: DBMAIL.SENDPAGE\n" );

    /* connect to database */
    EXEC SQL CONNECT TO sample;
    /* set error handler */
    EXEC SQL WHENEVER SQLERROR GOTO err_routine;

    /* deaclear cursor to fetch administator cell phone numbers
    and send alert messages */
    EXEC SQL DECLARE c1 CURSOR FOR
        SELECT replace(
            replace(
                replace(cell_phone, '(', ''),
                ')', ''),
            ' ', '')
        FROM staff WHERE job='Admin';

    /* open cursor */
    EXEC SQL OPEN c1;

    do {
        EXEC SQL FETCH c1 INTO :phone;
        if (SQLCODE != 0) break;

        /* execute DB Mail procedure */
        EXEC SQL CALL DBMAIL.SENDPAGE(:phone, 'Free space in the database is
            critically low. Your immediate
```

```
                                attention is required',
                                errors, &ret);
    if (ret<0) /* print error */
        printf ("DB Mail Error: %s \n ", errors);
    else
        printf ("Message sent to %s \n ", phone);

} while ( 1 );

EXEC SQL CLOSE c1;

/* reset connection */
EXEC SQL CONNECT RESET;
return 0;

err_routine:
    printf (" SQL Error, SQLCODE = %d \n ", SQLCODE);
    return -1;

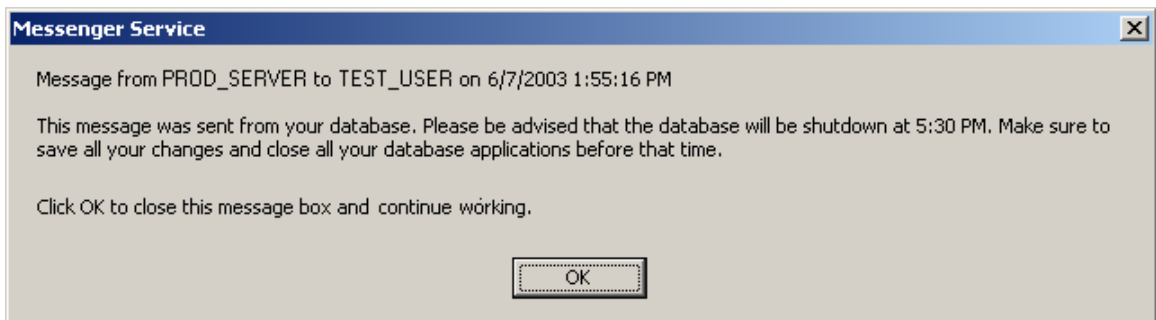
}
```

CHAPTER 8, Sending network popup messages

Overview

To send network popup messages from database applications use the `dbmail.SendPopupMessage` method. This method can be used to send messages to single or multiple message recipients or even to broadcast messages to all users in a domain or workgroup. In order to receive network messages users must be running Windows NT 4, Windows 2000, Windows XP or later and the standard Messenger service must be enabled.

Here is an example message as it appears on the user's Desktop.



Messages sent using the `dbmail.SendPopupMessage` are similar to messages sent using Window NT native NET SEND command.

For detailed method descriptions and usage examples specific to your database system refer to the following topics in this chapter.

Oracle

SEND_POPUP_MESSAGE

Use this method to send network popup messages from your database applications. The definition of SEND_POPUP_MESSAGE is shown below:

Definition

```
db_mail.send_popup_message(
    recipients      VARCHAR2,
    message         VARCHAR2 )
RETURN NUMBER
```

Argument	Max Size; Value Range	Description
Recipients	4000	A registered message alias such as network computer name or user name. Use comma to separate multiple recipient names.
Message	32765	Message text. Although message size can be up to 32765 characters long, it is not recommended to send messages that are longer than 1000 characters as the created pop up message box will take more screen space than can fit most Desktops. Because of this, the OK button may appear below the visible screen area thus making it difficult for the recipient to close the message box.

Return values:

0 - Success.

1 - Timed out. This procedure can timeout either because it cannot get a lock on the pipe, or because the pipe remains too full to be used.

3 - An interrupt occurred.

**Usage Tips:**

- If you add a call to the SEND_POPUP_MESSAGE function from a multi-row SQL statement such as SELECT ... FROM ... TABLE, Oracle will invoke the SEND_POPUP_MESSAGE function as many times as many rows are affected by the statement.
- You can configure DB Mail to broadcast messages to all users in a domain or workgroup. To use this feature you should enable the **Broadcast** option in DB Mail configuration. See [Configuring Network Popups and Alerts Options](#) topic for more information. To broadcast messages specify NULL for the **Recipients** parameters.

The following example demonstrates how to send popup messages using DB Mail functions.

Example:

The following SELECT statement will send broadcast message to all database users.

```
SELECT db_mail.send_popup_message(NULL,
  'This message was sent from your database. Please be advised that ' ||
  'the database will be shutdown at 5:30 PM. Make sure to save all ' ||
  'your changes and close all your database applications before that ' ||
  'time.' || chr(10) || chr(10) ||
  'Click OK to close this message box and continue working.' )
FROM dual;
```

Microsoft SQL Server

SendPopupMessage

Use this method to send network popup messages from your database applications. The definition of SendPopupMessage procedure is shown below:

Definition

```
PROCEDURE dbmail.SendPopupMessage(
    @recipients    VARCHAR(8000),
    @message       VARCHAR(1000) )
```

Argument	Max Size; Value Range	Description
Recipients	8000	A registered message alias such as network computer name or user name. Use comma to separate multiple recipient names.
Message	1000	Message text.

Return values: Returns unique message ID or returns -1 if an error occurs.



Usage Tips:

Use the @@ERROR global variable to check for errors. @@ERROR is set to 0 if the procedure executed successfully. If an error occurs, a non-zero number is returned. @@ERROR returns the number of the error message until another Transact-SQL statement is executed. You can view the text associated with an @@ERROR error number in the SYSMESSAGES system table.

The following example demonstrates how to send popup messages using DB Mail functions.

Example:

The following EXECUTE statement will send broadcast message to all database users.

```
EXEC master.dbmail.SendPopupMessage NULL,
    'This message was sent from your database. Please be advised that
    the database will be shutdown at 5:30 PM. Make sure to save all
    your changes and close all your database applications before that
    time.

    Click OK to close this message box and continue working.'
```

Sybase SQL Server, ASE, ASA

SendPopupMessage

Use this method to send network popup messages from your database applications. The definition of SendPopupMessage procedure is shown below:

Definition

```
PROCEDURE dbmail.SendPopupMessage(
    @recipients      VARCHAR(255),
    @message         VARCHAR(255),
    OUT @SQLMessage  VARCHAR(255),
    OUT @result      INT)
```

Argument	Max Size; Value Range	Description
Recipients	255	A registered message alias such as network computer name or user name. Use comma to separate multiple recipient names.
Message	255	Message text.
SQLMessage	255	Output variable used to return error description in case an error occurs
Result		Output variable to return procedure result

Return values:

Result returns unique message ID for the sent message (a positive number) if the procedure completes successfully.

If an error occurs, **Result** returns a negative number. Use **SQLMessage** argument to obtain the error description.

The following example demonstrates how to send popup messages using DB Mail functions.

Example:

The following EXECUTE statement will notify John Doe about meeting cancellation.

```
DECLARE @Ret INT, @ErrorMessage VARCHAR(255)
EXEC sybssystemprocs.dbmail.SendPopupMessage 'JohnDoe',
    'Today''s sales team meeting has been canceled.', @ErrorMessage, @Ret
```

IBM DB2

SendPopupMessage

Use this method to send network popup messages from your database applications. The definition of SendPopupMessage procedure is shown below:

Definition

```
PROCEDURE dbmail.SendPopupMessage(
    recipients      VARCHAR(32672),
```



```

message          VARCHAR(1000),
OUT SQLMessage  VARCHAR(1000),
OUT result      INT)

```

Argument	Max Size; Value Range	Description
Recipients	32672	A registered message alias such as network computer name or user name. Use comma to separate multiple recipient names.
Message	1000	Message text.
SQLMessage	1000	Output variable used to return error description in case an error occurs
Result		Output variable to return procedure result

Return values:

Result returns unique message ID for the sent message (a positive number) if the procedure completes successfully.

If an error occurs, **Result** returns a negative number. Use **SQLMessage** argument to obtain the error description.

The following example demonstrates how to send popup messages using DB Mail functions.

Example 1 (DB2 SQL, single message recipient):

The following CALL statement will notify John Doe about meeting cancellation.

```

DECLARE Ret INT;
DECLARE ErrorMessage VARCHAR(1000);

CALL dbmail.SendPopupMessage( 'JohnDoe',
    'Today''s sales team meeting has been canceled.', ErrorMessage, Ret);

```

Example 2 (DB2 SQL, message broadcast):

The following CALL statement will send broadcast message to all database users.

```

DECLARE Ret INT;
DECLARE ErrorMessage VARCHAR(1000);

CALL dbmail.SendPopupMessage( '',
    'This message was sent from your database. Please be advised that
    the database will be shutdown at 5:30 PM. Make sure to save all
    your changes and close all your database applications before that
    time.

    Click OK to close this message box and continue working.',
    ErrorMessage, Ret);

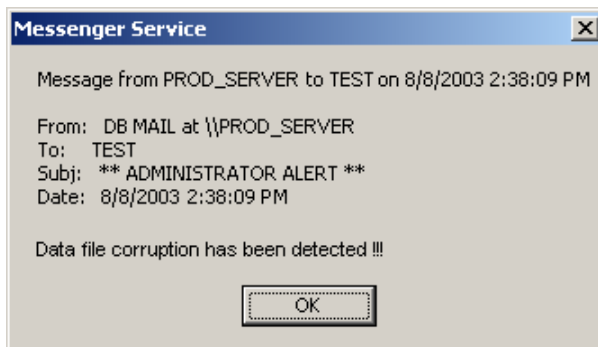
```

CHAPTER 9, Sending system alerts

Overview

To send system alerts from database applications use the *dbmail.SendAlert* method. This method can be used to send interruptible popup messages (administrative alerts) to all users whose name appear in the Window NT alert list. Essentially this message type is similar to sending network popup messages using the *dbmail.SendPopupMessage* method. The only real difference is that message recipients are managed outside of the database and can be changed without making any changes in the database. In order to receive alerts, users must be running Windows NT 4, Windows 2000, Windows XP or later and the standard Messenger service must be enabled.

Here is an example message as it appears on the user's Desktop.



Messages sent using the *dbmail.SendAlert* are similar to messages sent by the Window NT native Alerter service.

For detailed method descriptions and usage examples specific to your database system refer to the following topics in this chapter.

Oracle

SEND_ALERT

Use this method to send administrative alerts from your database applications. The definition of *SEND_ALERT* function is shown below:

Definition

```
db_mail.send_alert(
    message          VARCHAR2 )
RETURN NUMBER
```

Argument	Max Size; Value Range	Description
Message	32765	Message text. Although message size can be up to 32765 characters long, it is not recommended to send messages that are longer than 1000 characters as the created pop up message box will take more screen space than can fit most Desktops. Because of this, the OK button may appear below the visible screen area thus making it difficult for the recipient to close the message box.

Return values:

0 - Success.

1 - Timed out. This procedure can timeout either because it cannot get a lock on the pipe, or because the pipe remains too full to be used.

3 - An interrupt occurred.

The following example demonstrates how to send alert messages using DB Mail functions.

Example:

```
SELECT db_mail.send_alert('Database file corruption has been detected !!!' )
FROM dual;
```

Microsoft SQL Server

SendAlert

Use this method to send administrative alerts from your database applications. The definition of SendAlert procedure is shown below:

Definition

```
PROCEDURE dbmail.SendAlert(
    @message          VARCHAR(1000) )
```

Argument	Max Size; Value Range	Description
Message	1000	Message text.

Return values: Returns unique message ID or returns -1 if an error occurs.

 **Usage Tips:**

Use the @@ERROR global variable to check for errors. @@ERROR is set to 0 if the procedure executed successfully. If an error occurs, a non-zero number is returned. @@ERROR returns the number of the error message until another Transact-SQL statement is executed. You can view the text associated with an @@ERROR error number in the SYSMESSAGES system table.

Example:

The following EXECUTE statement will send administrative alert to all system administrators.

```
EXEC master.dbmail.SendAlert 'Database file corruption has been detected !!!'
```

Sybase SQL Server, ASE, ASA

SendAlert

Use this method to send administrative alerts from your database applications. The definition of SendAlert procedure is shown below:

Definition

```
PROCEDURE dbmail.SendAlert(
    @message          VARCHAR( 255 ),
    OUT @SQLMessage   VARCHAR( 255 ),
    OUT @result       INT)
```

Argument	Max Size; Value Range	Description
Message	255	Message text.
SQLMessage	255	Output variable used to return error description in case an error occurs
Result		Output variable to return procedure result

Return values:

Result returns unique message ID for the sent message (a positive number) if the procedure completes successfully.

If an error occurs, **Result** returns a negative number. Use **SQLMessage** argument to obtain the error description.

Example:

The following EXECUTE statement will send administrative alert to all system administrators.

```

DECLARE @Ret INT, @ErrorMessage VARCHAR(255)
EXEC sybssystemprocs.dbmail.SendAlert
    'Database file corruption has been detected !!!',
    @ErrorMessage, @Ret

```

IBM DB2

SendAlert

Use this method to send administrative alerts from your database applications. The definition of SendAlert procedure is shown below:

Definition

```

PROCEDURE dbmail.SendAlert(
    message          VARCHAR(1000),
    OUT SQLMessage  VARCHAR(1000),
    OUT result       INT)

```

Argument	Max Size; Value Range	Description
Message	1000	Message text.
SQLMessage	1000	Output variable used to return error description in case an error occurs
Result		Output variable to return procedure result

Return values:

Result returns unique message ID for the sent message (a positive number) if the procedure completes successfully.

If an error occurs, **Result** returns a negative number. Use **SQLMessage** argument to obtain the error description.

Example (DB2 SQL):

The following CALL statement will send administrative alert to all system administrators.

```

DECLARE Ret INT;
DECLARE ErrorMessage VARCHAR(1000);
CALL dbmail.SendAlert(
    'Database file corruption has been detected !!!',
    ErrorMessage, Ret);

```

CHAPTER 10, Sending electronic faxes

Overview

To send electronic faxes from database applications use the `dbmail.SendFax` or `dbmail.SendFaxEx` methods. Both methods allow you to specify a text-based file such as HTML or XML that DB Mail will convert to fax-compatible TIFF image. You can create files to be faxed either directly from your applications using file access methods available in the application programming system. Alternately, you can call the `dbmail.CreateMailFile` method. In any scenario, after such file is created you would call one of SendFax methods which converts the file to a TIFF image. DB Mail then communicates to the fax server and transmits the previously created TIFF image.

Why HTML and XML?

DB Mail internally uses Microsoft® Internet Explorer® control to render HTML and XML pages. Both HTML and XML provide ways to describe the visual appearance of the documents, yet this information is stored in flat text files that can be easily written from SQL and from various database applications. Using the latest versions of Internet Explorer and a wide variety of features available in the latest HTML and/or XML incarnations, you can create documents with virtually any layout and then use DB Mail to automatically fax such documents. For example, you can automate unattended creation and faxing of such common business documents as invoices, bill of ladings, transaction receipts, inventory reports and so on. This requires that you possess certain knowledge of HTML and/or XML features, which are not covered in this manual. A good place to learn HTML and Internet Explorer features is Microsoft Development Network. You can find complete HTML and DHTML reference at <http://msdn.microsoft.com/library/default.asp?url=/workshop/author/html/reference/elements.asp>

Advanced options

DB Mail provides options for attaching cover pages to faxed documents. Using these options you dynamically fill cover page properties including recipient name, company and fax, sender information, cover page message and so on. You can also instruct DB Mail to notify you by email after successful or failed fax transmissions, or even in both cases.

DB Mail supports multiple fax cover pages. When creating fax messages you can select which cover page you want to use. Different users and applications can use different cover pages.

For more information on how to create or edit cover pages, refer to [Creating and modifying cover pages](#) topic.

For detailed descriptions of the `dbmail.SendFax` method and usage examples specific to your database system refer to the following topics in this chapter.

Creating and modifying cover pages

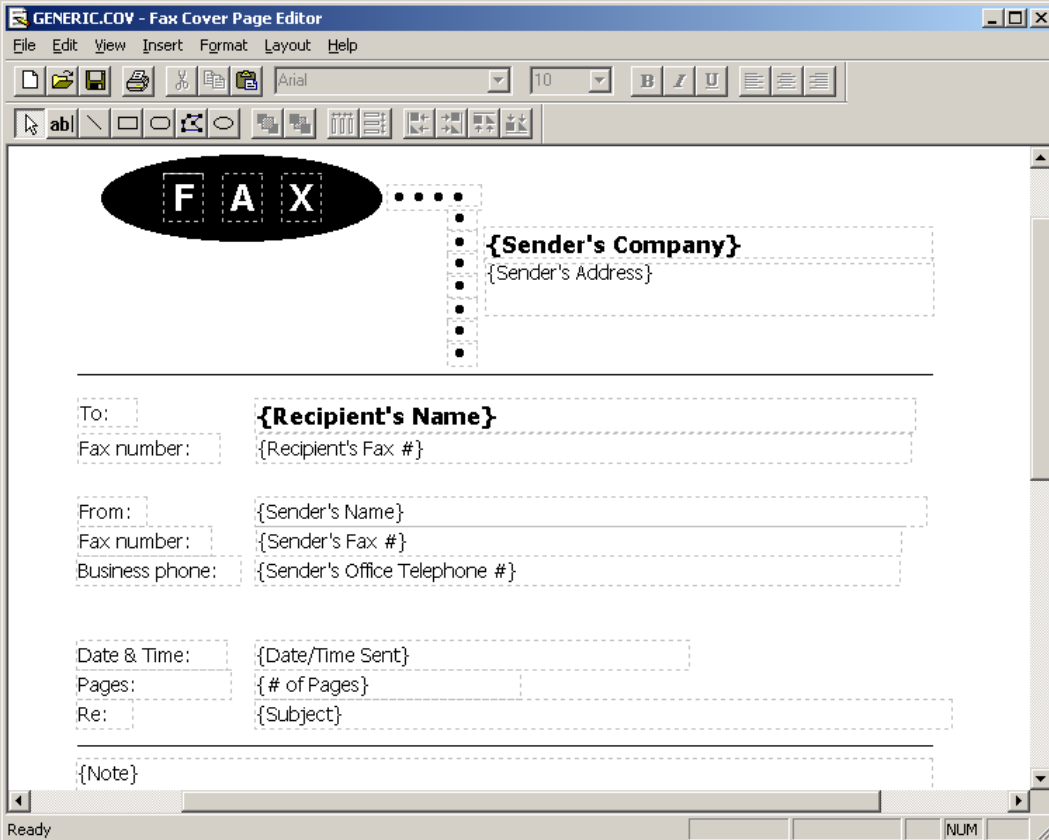
To create, add, edit, or delete cover pages ppen Fax applet in the Windows Control Panel. On the **Cover Pages** tab, do one or more of the following:

To	Do this
Create a new cover page	Click New to start the Fax Cover Page Editor.
Add a cover page to your personal cover pages list	Click Add to open the Browse for New Cover Page File dialog box. Locate the desired cover page, and then click Open to add the cover page file to your personal cover pages list. You only need to add a cover page if you have a personal cover page saved in another location other than the default directory.
Open and edit an existing cover page	Click a fax cover page to select it, and then click Open to start the Fax Cover Page Editor.
Delete an existing cover page	Click a fax cover page to select it, and then click Delete .

 **Important Notes:**

- To open **Fax**, click **Start**, point to **Settings**, click **Control Panel**, and then double-click **Fax**. If **Fax** does not appear, you need to install a fax device (such as a modem).
- Cover pages must have a .COV file extension. If you cannot find the cover page you are looking for, make sure it has the correct extension.

Sample cover page design is shown below



The screenshot shows the 'GENERIC.COV - Fax Cover Page Editor' window. The interface includes a menu bar (File, Edit, View, Insert, Format, Layout, Help), a toolbar with icons for file operations and formatting, and a main editing area. The cover page design features a large 'FAX' logo in a black oval at the top left. To its right is a dashed box for the sender's company name and address. Below this is a horizontal line. The recipient information section includes fields for 'To:' (Recipient's Name), 'Fax number:' (Recipient's Fax #), 'From:' (Sender's Name), 'Fax number:' (Sender's Fax #), and 'Business phone:' (Sender's Office Telephone #). Further down are fields for 'Date & Time:' (Date/Time Sent), 'Pages:' (# of Pages), and 'Re:' (Subject). At the bottom is a large dashed box for a note. The status bar at the bottom left shows 'Ready' and the bottom right shows 'NUM'.

Oracle

SEND_FAX

Use this method to fax documents directly from your database or from external database-connected applications. The definition of the SEND_FAX function is shown below:

Definition

```
db_mail.send_fax(
    fax                VARCHAR2,
    subject            VARCHAR2,
    file_name          VARCHAR2,
    cover_page_name    VARCHAR2 DEFAULT NULL,
    recipient_name     VARCHAR2 DEFAULT NULL)
RETURN NUMBER
```

Argument	Max Size; Value Range	Description
Fax	50	Recipient's fax number. Specify the number exactly as you would dial it from your phone number including any dial out numbers, long distance codes, local area codes and so on. Digits in the number can be separated by optional dashes, parenthesis, spaces and other non-digit symbols. For example, 1 (212) 555-6789
Subject	255	Message subject. If Cover_page_name value is not NULL and the cover page features the RE field, the subject text is used for that field. See sample cover page in Creating and modifying cover pages section for details.
File_Name	255	Name of the file to fax. Name must conform to standard file naming conventions and should not include file path. The referenced file must exist in the directory specified by the MAILSTORE directory object.
Cover_page_name	50	Cover page file name not including file path and file extension. If NULL value is passed for the Cover_page_name argument, no cover page is attached to the fax.
Recipient_name	50	Recipient's name

Return values:

0 - Success.

1 - Timed out. This procedure can timeout either because it cannot get a lock on the pipe, or because the pipe remains too full to be used.

3 - An interrupt occurred.

An exception –20021 is raised in case the file cannot be found or loaded. Use Oracle SQLERRM session variable to obtain the error description.

Usage Tips:

- If you add a call to the SEND_FAX function from a multi-row SQL statement such as SELECT ... FROM ... TABLE, Oracle will invoke the SEND_FAX function as many times as many rows are affected by the statement.
- HTML and XML files used for faxing can refer to other files such as images, cascading style sheets, and so on as long as these files are accessible via the Intranet or Internet from the computer running DB Mail Server. The composite document is rendered using all referenced objects at the time the message is processed and converted to a single TIFF image.
- When developing your fax procedures, you can use Internet Explorer or other web browser to preview and test the created documents. What you see in the browser Print Preview mode is what you will get in the fax. Faxing HTML and XML documents is virtually the same as printing them to a printer. The only real difference is that instead of producing printed hard copies DB Mail produces TIFF images and then automatically faxes them to the destination fax number.

Example:

The following example demonstrates how to send electronic fax using DB Mail functions.

In this example we will dynamically create a HTML report and then fax it.

```

DECLARE
  ret_code INTEGER;
BEGIN
  db_mail.create_mail_file('report.htm',
    '<html><title>Test Report</title><body>' || chr(10) ||
    '<h2>Test Report</h2><hr>' || chr(10) ||
    '<table><tr><th bgcolor=black><font color=white>Col 1</th>' ||
    '<th bgcolor=black><font color=white>Col 2</th></tr>' || chr(10) ||
    '<tr><td>Value A</td><td align=right>1</td></tr>' || chr(10) ||
    '<tr><td>Value B</td><td align=right>2</td></tr>' || chr(10) ||
    '</table></body></html>', FALSE );

  ret_code := db_mail.send_fax('+1 (123) 222-3344',
    'Test fax subject',
    'report.htm',
    'Generic',
    'Test recipient');

  db_mail.delete_mail_file('report.htm');

  IF ret_code != 0 THEN
    raise_application_error(-20010, 'SEND_FAX Status = ' || ret_code);
  END IF;
END;
```

SEND_FAX_EX

Use this method to fax documents directly from your database or from external database-connected applications. SEND_FAX_EX is an extended version of SEND_FAX procedure. Simply put SEND_FAX_EX supports more options than SEND_FAX function. The definition of SEND_FAX_EX function is shown below:

Definition

```
db_mail.send_fax_ex(
    fax                VARCHAR2,
    subject            VARCHAR2,
    file_name          VARCHAR2,
    cover_page_name    VARCHAR2 DEFAULT NULL,
    recipient_name     VARCHAR2 DEFAULT NULL,
    recipient_co       VARCHAR2 DEFAULT NULL,
    recipient_phone    VARCHAR2 DEFAULT NULL,
    cover_page_message VARCHAR2 DEFAULT NULL,
    sender_name        VARCHAR2 DEFAULT NULL,
    sender_co          VARCHAR2 DEFAULT NULL,
    sender_dept        VARCHAR2 DEFAULT NULL,
    sender_phone       VARCHAR2 DEFAULT NULL,
    sender_fax         VARCHAR2 DEFAULT NULL,
    notify_on_success  BOOLEAN DEFAULT FALSE,
    notify_on_failure  BOOLEAN DEFAULT FALSE,
    notify_email       VARCHAR2 DEFAULT NULL,
    priority           INTEGER DEFAULT 1 )
RETURN NUMBER
```

Argument	Max Size; Value Range	Description
Fax	50	Recipient's fax number. Specify the number exactly as you would dial it from your phone number including any dial out numbers, long distance codes, local area codes and so on. Digits in the number can be separated by optional dashes, parenthesis, spaces and other non-digit symbols. For example, 1 (212) 555-6789
Subject	255	Message subject. If Cover_page_name value is not NULL and the cover page features the RE field, the subject text is used for that field. See sample cover page in Creating and modifying cover pages section for details.
File_Name	255	Name of the file to fax. The name must conform to standard file naming conventions and should not include file path. The referenced file must exist in the directory specified by the MAILSTORE directory object.
Cover_page_name	50	Cover page file name not including file path and file extension. . If NULL value is passed for the Cover_page_name argument, no cover page is attached to the fax.
Recipient_name	50	Recipient's name.
Recipient_co	100	Recipients' company name, specify NULL if not available.
Recipient_phone	50	Recipient phone number. The phone number can be specified in any format and may also include phone extensions if any. For example (121) 555-3456 x123

Cover_page_message	1000	Free message text that will appear in the cover page message area.
Sender_name	50	Sender's name.
Sender_co	100	Sender's company name
Sender_dept	100	Sender's department, for example <i>Sales</i> .
Sender_phone	50	Sender's phone number. The phone number can be specified in any format and may also include phone extensions if any. For example (121) 098-3456 x123
Sender_fax	50	Sender's fax number. The fax number can be specified in any format.
Notify_on_success	TRUE/FALSE	Whether to notify message sender by email after fax transmission completed successfully. If TRUE, in case of a successful transmission DB Mail will email notification to the email address specified in Notify_Email argument.
Notify_on_failure	TRUE/FALSE	Whether to notify message sender by email after fax transmission failed. If TRUE, in case of a failed transmission DB Mail will email notification to the email address specified in Notify_Email argument.
Notify_Email	100	Email address of the person or email group that will receive email notification in case Notify_on_success or Notify_on_failure value is TRUE.

Return values:

0 - Success.

1 - Timed out. This procedure can timeout either because it cannot get a lock on the pipe, or because the pipe remains too full to be used.

3 - An interrupt occurred.

An exception –20021 is raised in case the file cannot be found or loaded. Use Oracle SQLERRM session variable to obtain the error description.

**Usage Tips:**

- If you add a call to the SEND_FAX function from a multi-row SQL statement such as SELECT ... FROM ... TABLE, Oracle will invoke the SEND_FAX function as many times as many rows are affected by the statement.
- HTML and XML files used for faxing can refer to other files such as images, cascading style sheets, and so on as long as these files are accessible via the Intranet or Internet from the computer running DB Mail Server. The composite document is rendered using all referenced objects at the time the message is processed and converted to a single TIFF image.
- When developing your fax procedures, you can use Internet Explorer or other web browser to preview and test the created documents. What you see in the browser Print Preview

mode is what you get in the fax. Faxing HTML and XML documents is virtually the same as printing them to a printer. The only real difference is that instead of producing printed hard-copies DB Mail produces TIFF images and then automatically faxes them to the destination fax number.

Example:

In this example we will dynamically create invoices for all orders placed yesterday and then fax them to customers.

```

DECLARE
    invoice_html VARCHAR2(4000);
BEGIN

    -- for every invoice in the orders table with yesterday's date
    -- build invoice document
    FOR rec IN (SELECT * FROM orders WHERE order_date = trunc(sysdate)-1)
    LOOP
        -- make invoice header
        invoice_html :=
            '<html><head>' ||
                '<title>Invoice #' || rec.order_no || '</title>' ||
                '<link rel=stylesheet href="http://www.company.com/style.css">' ||
            '</head>' ||
            '<body>' ||
                '<h2>Invoice #' || rec.order_no || '</h2><hr>' ||
                '<table>' ||
                    '<tr><th class=inv_header>Order Date</th>' ||
                    '<th class=inv_header>PO #</th>' ||
                    '<th class=inv_header>Customer #</th>' ||
                    '<th class=inv_header>Customer Name</th>' ||
                    '<th class=inv_header>Ship Via</th>' ||
                    '<th class=inv_header>Ship Date</th>' ||
                '</tr>' ||
                '<tr><td class=inv_data>' || rec.order_date || '</td>' ||
                '<td class=inv_data>' || rec.po_no || '</td>' ||
                '<td class=inv_data>' || rec.cust_no || '</td>' ||
                '<td class=inv_data>' || rec.cust_name || '</td>' ||
                '<td class=inv_data>' || rec.ship_via || '</td>' ||
                '<td class=inv_data>' || rec.ship_date || '</td>' ||
                '</tr>' ||
            '</table>';
        db_mail.create_mail_file('invoice.htm', invoice_html, FALSE );

        -- now add invoice details
        invoice_html :=
            '<table>' ||
                '<tr><th class=inv_header>Line #</th>' ||
                '<th class=inv_header>Item Code</th>' ||
                '<th class=inv_header>Item Name</th>' ||
                '<th class=inv_header>Quantity</th>' ||
                '<th class=inv_header>Price</th>' ||
            '</tr>';
        db_mail.create_mail_file('invoice.htm', invoice_html, TRUE );

        FOR rec2 IN (SELECT * FROM order_items WHERE order_no = rec.order_no)
        LOOP
            invoice_html :=
                '<tr><td class=inv_data>' || rec2.line_no || '</td>' ||
                '<td class=inv_data>' || rec2.item || '</td>' ||
                '<td class=inv_data>' || rec2.item_name || '</td>' ||

```

```

        '<td class=inv_data>' || rec2.quantity || '</td>' ||
        '<td class=inv_data>' || rec2.price || '</td>' ||
    </tr>';
    db_mail.create_mail_file('invoice.htm', invoice_html, TRUE );
END LOOP;

-- now add invoice total and save it as a file
invoice_html := '</table>' ||
    '<hr>' ||
    '<p class=totals><h3>TOTAL: ' || rec.order_amount || '</p>' ||
    '</body>' ||
    '</html>';
dbdb_mail.create_mail_file('invoice.htm', invoice_html, TRUE );

-- fax invoice
ret_code := db_mail.send_fax_ex(
    rec.cust_fax, 'Invoice #' || rec.order_no,
    'invoice.htm', 'InvoiceCoverPage',
    rec.cust_name, rec.cust_company,
    rec.cust_phone, NULL,
    'This is your invoice. Call our sales department ' ||
    'at (111) 222-3344 if you have any questions ' ||
    'concerning this invoice',
    NULL, 'My company', 'Sales department',
    '(111) 222-3344', '(111) 222-3355',
    FALSE, TRUE, 'ar@company.com', 0 );
-- move to the next order and create/fax next invoice
END LOOP;
END;
```

Microsoft SQL Server

SendFax

Use this method to fax documents directly from your database or from external database-connected applications. The definition of SendFax function is shown below:

Definition

```

dbmail.SendFax(
    @fax                VARCHAR(50),
    @subject            VARCHAR(255),
    @file_name         VARCHAR(255),
    @cover_page_name   VARCHAR(50) = NULL,
    @recipient_name    VARCHAR(50) = NULL)
```

Argument	Max Size; Value Range	Description
Fax	50	Recipient's fax number. Specify the number exactly as you would dial it from your phone number including any dial out numbers, long distance codes, local area codes and so on. Digits in the number can be separated by optional dashes, parenthesis, spaces and other non-digit symbols. For example, 1 (212) 555-6789
Subject	255	Message subject. If Cover_page_name value is not NULL and the cover page features the RE field, the subject text is used for that field. See

		sample cover page in Creating and modifying cover pages section for details.
File_Name	255	Name of the file to fax. The name must conform to standard file naming conventions and may include the file path.
Cover_page_name	50	Cover page file name not including file path and file extension. If NULL value is passed for the Cover_page_name argument, no cover page is attached to the fax.
Recipient_name	50	Recipient's name

Return values: Returns unique message ID or returns -1 if an error occurs.

Usage Tips:

- Use the @@ERROR global variable to check for errors. @@ERROR is set to 0 if the procedure executed successfully. If an error occurs, a non-zero number is returned. @@ERROR returns the number of the error message until another Transact-SQL statement is executed. You can view the text associated with an @@ERROR error number in the SYSMESSAGES system table.
- HTML and XML files used for faxing can refer to other files such as images, cascading style sheets, and so on as long as these files are accessible via the Intranet or Internet from the computer running DB Mail Server. The composite document is rendered using all referenced objects at the time the message is processed and converted to a single TIFF image.
- When developing your fax procedures, you can use Internet Explorer or other web browser to preview and test the created documents. What you see in the browser Print Preview mode is what you will get in the fax. Faxing HTML and XML documents is virtually the same as printing them to a printer. The only real difference is that instead of producing printed hard copies DB Mail produces TIFF images and then automatically faxes them to the destination fax number.

Example:

The following example demonstrates how to send electronic fax using DB Mail functions.

In this example we will dynamically create HTML report and then fax it.

```
EXEC master.dbmail.CreateMailFile 'c:\temp\report.htm',
    '<html><title>Test Report</title><body>
    <h2>Test Report</h2><hr>
    <table><tr><th bgcolor=black><font color=white>Col 1</th>
    <th bgcolor=black><font color=white>Col 2</th></tr>
    <tr><td>Value A</td><td align=right>1</td></tr>
    <tr><td>Value B</td><td align=right>2</td></tr>
    </table></body></html>', 0

EXEC master.dbmail.SendFax '+1 (123) 222-3344',
    'Test fax subject',
    'c:\temp\report.htm',
    'Generic',
    'Test recipient'

EXEC master.dbmail.DeleteMailFile 'c:\temp\report.htm'
```

SendFaxEx

Use this method to fax documents directly from your database or from external database-connected applications. SendFaxEx is an extended version of SendFax procedure. Simply put SendFaxEx supports more options than SendFax function. The definition of SendFaxEx procedure is shown below:

Definition

```
db_mail.SendFaxEx(
    @fax                VARCHAR( 50 ),
    @subject            VARCHAR( 255 ),
    @file_name         VARCHAR( 255 ),
    @cover_page_name   VARCHAR( 50 ) = NULL,
    @recipient_name    VARCHAR( 50 ) = NULL,
    @recipient_co      VARCHAR( 50 ) = NULL,
    @recipient_phone   VARCHAR( 50 ) = NULL,
    @cover_page_message VARCHAR( 1000 ) = NULL,
    @sender_name       VARCHAR( 50 ) = NULL,
    @sender_co        VARCHAR( 50 ) = NULL,
    @sender_dept      VARCHAR( 50 ) = NULL,
    @sender_phone     VARCHAR( 50 ) = NULL,
    @sender_fax       VARCHAR( 50 ) = NULL,
    @notify_on_success SMALLINT = 0,
    @notify_on_failure SMALLINT = 0,
    @notify_email     VARCHAR( 100 ) = NULL,
    @priority         INTEGER = 0 )
RETURN NUMBER
```

Argument	Max Size; Value Range	Description
Fax	50	Recipient's fax number. Specify the number exactly as you would dial it from your phone number including any dial out numbers, long distance codes, local area codes and so on. Digits in the number can be separated by optional dashes, parenthesis, spaces and other non-digit symbols. For example, 1 (212) 555-6789
Subject	255	Message subject. If Cover_page_name value is not NULL and the cover page features the RE field, the subject text is used for that field. See sample cover page in Creating and modifying cover pages section for details.
File_Name	255	Name of the file to fax. The name must conform to standard file naming conventions and may include file path.
Cover_page_name	50	Cover page file name not including file path and file extension. If NULL value is passed for the Cover_page_name argument, no cover page is attached to the fax.
Recipient_name	50	Recipient's name.
Recipient_co	50	Recipients' company name, specify NULL if not available.
Recipient_phone	50	Recipient phone number. The phone number can be specified in any format and may also include phone extensions if any. For example (121)

		555-3456 x123
Cover_page_message	1000	Free message text that will appear in the cover page message area.
Sender_name	50	Sender's name.
Sender_co	50	Sender's company name
Sender_dept	50	Sender's department, for example <i>Sales</i> .
Sender_phone	50	Sender's phone number. The phone number can be specified in any format and may also include phone extensions if any. For example (121) 555-3456 x123
Sender_fax	50	Sender's fax number. The fax number can be specified in any format.
Notify_on_success	0..1	Whether to notify message sender by email after fax transmission completed successfully. If this value is 1, in case of a successful transmission DB Mail will email notification to the email address specified in Notify_Email argument.
Notify_on_failure	0..1	Whether to notify message sender by email after fax transmission failed. If this value is 1, in case of a failed transmission DB Mail will email notification to the email address specified in Notify_Email argument.
Notify_Email	100	Email address of the person or email group that will receive email notification in case Notify_on_success or Notify_on_failure value is 1.

Return values: Returns unique message ID or returns -1 if an error occurs.

Usage Tips:

- Use the @@ERROR global variable to check for errors. @@ERROR is set to 0 if the procedure executed successfully. If an error occurs, a non-zero number is returned. @@ERROR returns the number of the error message until another Transact-SQL statement is executed. You can view the text associated with an @@ERROR error number in the SYSMESSAGES system table.
- HTML and XML files used for faxing can refer to other files such as images, cascading style sheets, and so on as long as these files are accessible via the Intranet or Internet from the computer running DB Mail Server. The composite document is rendered using all referenced objects at the time the message is processed and converted to a single TIFF image.
- When developing your fax procedures, you can use Internet Explorer or other web browser to preview and test the created documents. What you see in the browser Print Preview mode is what you will get in the fax. Faxing HTML and XML documents is virtually the same as printing them to a printer. The only real difference is that instead of producing printed hard copies DB Mail produces TIFF images and then automatically faxes them to the destination fax number.

Example:

In this example we will dynamically create invoices for all orders placed yesterday and then fax them to

customers.

```

DECLARE @invoice_html VARCHAR(8000)
DECLARE @order_no INT, @order_date DATETIME, @po_num VARCHAR(20),
        @cust_no INT, @cust_name VARCHAR(50), @ship_via CHAR(4),
        @ship_date DATETIME, @order_amount MONEY,
        @cust_fax VARCHAR(12), @cust_phone VARCHAR(12),
        @cust_company VARCHAR(50)
DECLARE @line_no INT, @item VARCHAR(10), @item_name VARCHAR(100),
        @quantity FLOAT, @price MONEY

DECLARE c1 CURSOR
SELECT order_no, order_date, po_num, cust_no,
       cust_name, ship_via, ship_date, order_amount,
       cust_fax, cust_phone, cust_company
FROM orders
WHERE order_date = DateAdd(day, -1, convert(datetime,
                                           convert(GetDate(), 101)))

-- build invoice document for every invoice
-- in the orders table with yesterday's date
OPEN c1
-- get first order
FETCH NEXT FROM c1 INTO @order_no, @order_date, @po_num,
                        @cust_no, @cust_name, @ship_via, @ship_date, @order_amount,
                        @cust_fax, @cust_phone, @cust_company

WHILE (@@fetch_status = 0)
BEGIN
    -- make invoice header
    SET @invoice_html =
        '<html><head>' +
        ' <title>Invoice #' + convert(varchar, @order_no) + '</title>' +
        ' <link rel=stylesheet href="http://www.company.com/style.css">' +
        '</head>' +
        '<body>' +
        ' <h2>Invoice #' + convert(varchar, @order_no) + '</h2><hr>' +
        '<table>' +
        ' <tr><th class=inv_header>Order Date</th>' +
        ' <th class=inv_header>PO #</th>' +
        ' <th class=inv_header>Customer #</th>' +
        ' <th class=inv_header>Customer Name</th>' +
        ' <th class=inv_header>Ship Via</th>' +
        ' <th class=inv_header>Ship Date</th>' +
        '</tr>' +
        ' <tr><td class=inv_data>' +
        '     convert(varchar, @order_date, 101) + '</td>' +
        ' <td class=inv_data>' +
        '     convert(varchar, @po_no) + '</td>' +
        ' <td class=inv_data>' +
        '     convert(varchar, @cust_no) + '</td>' +
        ' <td class=inv_data>' + @cust_name + '</td>' +
        ' <td class=inv_data>' + @ship_via + '</td>' +
        ' <td class=inv_data>' +
        '     convert(varchar, @ship_date, 101) + '</td>' +
        '</tr>' +
        '</table>'
    EXEC master.dbmail.CreateMailFile 'c:\temp\invoice.htm',
                                       @invoice_html, 0

    -- now add invoice details
    SET invoice_html =

```

```

        '<table>' +
        '<tr><th class=inv_header>Line #</th>' +
        '<th class=inv_header>Item Code</th>' +
        '<th class=inv_header>Item Name</th>' +
        '<th class=inv_header>Quantity</th>' +
        '<th class=inv_header>Price</th>' +
        '</tr>'
EXEC master.dbmail.CreateMailFile 'c:\temp\invoice.htm',
    @invoice_html, 1

-- declare internal cursor to fetch ordered items
DECLARE c2 CURSOR
SELECT line_no, item, item_name, quantity, price
FROM order_items
WHERE order_no = @order_no

OPEN c2
-- get first line
FETCH NEXT FROM c2 INTO @line_no, @item, @item_name,
    @quantity, @price

WHILE (@@fetch_status = 0)
BEGIN
    SET invoice_html =
        '<tr><td class=inv_data>' +
            convert(varchar, line_no) + '</td>' +
        '<td class=inv_data>' + @item + '</td>' +
        '<td class=inv_data>' + @item_name + '</td>' +
        '<td class=inv_data>' +
            convert(varchar, @quantity) + '</td>' +
        '<td class=inv_data>' +
            convert(varchar, price) + '</td>' +
        '</tr>'
    EXEC master.dbmail.CreateMailFile 'c:\temp\invoice.htm',
        @invoice_html, 1

    -- move to the next item
    FETCH NEXT FROM c2 INTO @line_no, @item, @item_name,
        @quantity, @price
END

CLOSE c2
DEALLOCATE c2

-- now add invoice total and also save it as a file
SET invoice_html =
    '</table>' +
    '<hr>' +
    '<p class=totals><h3>TOTAL: ' +
        convert(varchar, @order_amount) + '</p>' +
    '</body>' +
    '</html>'
EXEC master.dbmail.CreateMailFile 'c:\temp\invoice.htm',
    @invoice_html, 1

-- fax invoice
EXEC master.dbmail.SendFaxEx
    @cust_fax, 'Invoice',
    'c:\temp\invoice.htm', 'InvoiceCoverPage',
    @cust_name, @cust_company,
    @cust_phone, NULL,
    'This is your invoice. Call our sales department
    at (111) 222-3344 if you have any questions
    concerning this invoice',

```

```

NULL, 'My company', 'Sales department',
'(111) 222-3344', '(111) 222-3355',
0, 1, 'ar@company.com', 0

-- move to the next order and create/fax next invoice
FETCH NEXT FROM c1 INTO @order_no, @order_date, @po_num,
    @cust_no, @cust_name, @ship_via, @ship_date, @order_amount,
    @cust_fax, @cust_phone, @cust_company

END

CLOSE c1
DEALLOCATE c1

```

Sybase SQL Server, ASE, ASA

SendFax

Use this method to fax documents directly from your database or from external database-connected applications. The definition of SendFax function is shown below:

Definition

```

dbmail.SendFax(
    @fax                VARCHAR(50),
    @subject            VARCHAR(255),
    @file_name         VARCHAR(255),
    @cover_page_name   VARCHAR(50),
    @recipient_name    VARCHAR(50),
    OUT @SQLMessage    VARCHAR(255),
    OUT @result        INT)

```

Argument	Max Size; Value Range	Description
Fax	50	Recipient's fax number. Specify the number exactly as you would dial it from your phone number including any dial out numbers, long distance codes, local area codes and so on. Digits in the number can be separated by optional dashes, parenthesis, spaces and other non-digit symbols. For example, 1 (212) 555-6789
Subject	255	Message subject. If Cover_page_name value is not NULL and the cover page features the RE field, the subject text is used for that field. See sample cover page in Creating and modifying cover pages section for details.
File_Name	255	Name of the file to fax. Name must conform to standard file naming conventions and should not include file path.
Cover_page_name	50	Cover page file name not including file path and file extension. If NULL value is passed for the Cover_page_name argument, no cover page is attached to the fax.
Recipient_name	50	Recipient's name

SQLMessage	255	Output variable used to return error description in case an error occurs
Result		Output variable to return procedure result

Return values:

Result returns unique message ID for the sent message (a positive number) if the procedure completes successfully.

If an error occurs, **Result** returns a negative number. Use **SQLMessage** argument to obtain the error description.

 **Usage Tips:**

- HTML and XML files used for faxing can refer to other files such as images, cascading style sheets, and so on as long as these files are accessible via the Intranet or Internet from the computer running DB Mail Server. The composite document is rendered using all referenced objects at the time the message is processed and converted to a single TIFF image.
- When developing your fax procedures you can use Internet Explorer or other web browser to preview and test the created documents. What you see in the browser Print Preview mode is what you will get in the fax. Faxing HTML and XML documents is virtually the same as printing them to a printer. The only real difference is that instead of producing printed hard copies DB Mail produces TIFF images and then automatically faxes them to the destination fax number.

Example:

The following example demonstrates how to send electronic fax using DB Mail functions.

In this example we will dynamically create HTML report and then fax it.

```

DECLARE @Ret INT, @ErrorMessage VARCHAR(255)
EXEC sybssystemprocs.dbmail.CreateMailFile 'report.htm',
    '<html><title>Test Report</title><body>
    <h2>Test Report</h2><hr>
    <table><tr><th bgcolor=black><font color=white>Col 1</th>
    <th bgcolor=black><font color=white>Col 2</th></tr>
    <tr><td>Value A</td><td align=right>1</td></tr>
    <tr><td>Value B</td><td align=right>2</td></tr>
    </table></body></html>', 0, @ErrorMessage, @Ret

EXEC sybssystemprocs.dbmail.SendFax '+1 (123) 222-3344',
    'Test fax subject',
    report.htm',
    'Generic',
    'Test recipient',
    @ErrorMessage, @Ret

EXEC sybssystemprocs.dbmail.DeleteMailFile 'report.htm', @ErrorMessage, @Ret

```

SendFaxEx

Use this method to fax documents directly from your database or from external database-connected applications. SendFaxEx is an extended version of SendFax procedure. Simply put SendFaxEx supports more options than SendFax function. The definition of SendFaxEx procedure is shown below:

Definition

```
db_mail.SendFaxEx(
    @fax          VARCHAR( 50 ),
    @subject     VARCHAR( 255 ),
    @file_name   VARCHAR( 255 ),
    @cover_page_name VARCHAR( 50 ),
    @recipient_name VARCHAR( 50 ),
    @recipient_co VARCHAR( 50 ),
    @recipient_phone VARCHAR( 50 ),
    @cover_page_message VARCHAR( 1000 ),
    @sender_name VARCHAR( 50 ),
    @sender_co   VARCHAR( 50 ),
    @sender_dept VARCHAR( 50 ),
    @sender_phone VARCHAR( 50 ),
    @sender_fax  VARCHAR( 50 ),
    @notify_on_success SMALLINT,
    @notify_on_failure SMALLINT,
    @notify_email VARCHAR( 100 ),
    @priority     INTEGER,
    OUT @SQLMessage VARCHAR( 255 ),
    OUT @result   INT )
RETURN NUMBER
```

Argument	Max Size; Value Range	Description
Fax	50	Recipient's fax number. Specify the number exactly as you would dial it from your phone number including any dial out numbers, long distance codes, local area codes and so on. Digits in the number can be separated by optional dashes, parenthesis, spaces and other non-digit symbols. For example, 1 (212) 555-6789
Subject	255	Message subject. If Cover_page_name value is not NULL and the cover page features the RE field, the subject text is used for that field. See sample cover page in Creating and modifying cover pages section for details.
File_Name	255	Name of the file to fax. Name must conform to standard file naming conventions and should not include file path.
Cover_page_name	50	Cover page file name not including file path and file extension.. If NULL value is passed for the Cover_page_name , argument, no cover page is attached to the fax.
Recipient_name	50	Recipient's name.
Recipient_co	50	Recipients' company name, specify NULL if not available.
Recipient_phone	50	Recipient phone number. The phone number can be specified in any format and may also include phone extensions if any. For example (121) 555-3456 x123

Cover_page_message	255	Free message text that will appear in the cover page message area.
Sender_name	50	Sender's name.
Sender_co	50	Sender's company name
Sender_dept	50	Sender's department, for example <i>Sales</i> .
Sender_phone	50	Sender's phone number. The phone number can be specified in any format and may also include phone extensions if any. For example (121) 555-3456 x123
Sender_fax	50	Sender's fax number. The fax number can be specified in any format.
Notify_on_success	0..1	Whether to notify message sender by email after fax transmission completed successfully. If this value is 1, in case of a successful transmission DB Mail will email notification to the email address specified in Notify_Email argument.
Notify_on_failure	0..1	Whether to notify message sender by email after fax transmission failed. If this value is 1, in case of a failed transmission DB Mail will email notification to the email address specified in Notify_Email argument.
Notify_Email	100	Email address of the person or email group that will receive email notification in case Notify_on_success or Notify_on_failure value is 1.
SQLMessage	255	Output variable used to return error description in case an error occurs
Result		Output variable to return procedure result

Return values:

Result returns unique message ID for the sent message (a positive number) if the procedure completes successfully.

If an error occurs, **Result** returns a negative number. Use **SQLMessage** argument to obtain the error description.

 **Usage Tips:**

- HTML and XML files used for faxing can refer to other files such as images, cascading style sheets, and so on as long as these files are accessible via the Intranet or Internet from the computer running DB Mail Server. The composite document is rendered using all referenced objects at the time the message is processed and converted to a single TIFF image.
- When developing your fax procedures you can use, Internet Explorer or other web browser to preview and test the created documents. What you see in the browser Print Preview mode is what you will get in the fax. Faxing HTML and XML documents is virtually the same as printing them to a printer. The only real difference is that instead of producing printed hard copies, DB Mail produces TIFF images and then automatically faxes them to the destination fax number.

Example:

In this example we will dynamically create invoices for all orders placed yesterday and then fax them to customers.

```

DECLARE @Ret INT, @ErrorMessage VARCHAR(255)
DECLARE @invoice_html VARCHAR(255)
DECLARE @order_no INT, @order_date DATETIME, @po_num VARCHAR(20),
        @cust_no INT, @cust_name VARCHAR(50), @ship_via CHAR(4),
        @ship_date DATETIME, @order_amount MONEY,
        @cust_fax VARCHAR(12), @cust_phone VARCHAR(12),
        @cust_company VARCHAR(50)
DECLARE @line_no INT, @item VARCHAR(10), @item_name VARCHAR(100),
        @quantity FLOAT, @price MONEY

DECLARE c1 CURSOR
SELECT order_no, order_date, po_num, cust_no,
        cust_name, ship_via, ship_date, order_amount,
        cust_fax, cust_phone, cust_company
FROM orders
WHERE order_date = DateAdd(day, -1, convert(datetime,
                                           convert(GetDate(), 101)))

-- build invoice document for every invoice
-- in the orders table with yesterday's date
OPEN c1
-- get first order
FETCH c1 INTO @order_no, @order_date, @po_num, @cust_no, @cust_name,
             @ship_via, @ship_date, @order_amount, @cust_fax, @cust_phone,
             @cust_company

WHILE (@@sqlstatus = 0)
BEGIN
    -- make invoice header
    SELECT @invoice_html =
        '<html><head>' +
        '<title>Invoice #' + convert(varchar, @order_no) + '</title>' +
        '<link rel=stylesheet href="http://www.company.com/style.css">' +
        '</head>'
    EXEC sybsystemprocs.dbmail.CreateMailFile 'invoice.htm', @invoice_html,
        0, @ErrorMessage, @Ret

    SELECT @invoice_html =
        '<body>' +
        '<h2>Invoice #' + convert(varchar, @order_no) + '</h2><hr>' +
        '<table>' +
        '<tr><th class=inv_header>Order Date</th>' +
        '<th class=inv_header>PO #</th>' +
        '<th class=inv_header>Customer #</th>' +
        '<th class=inv_header>Customer Name</th>' +
        '<th class=inv_header>Ship Via</th>' +
        '<th class=inv_header>Ship Date</th>' +
        '</tr>'
    EXEC sybsystemprocs.dbmail.CreateMailFile 'invoice.htm', @invoice_html,
        1, @ErrorMessage, @Ret

    SELECT @invoice_html =
        '<tr><td class=inv_data>' +
        convert(varchar, @order_date, 101) + '</td>' +
        '<td class=inv_data>' +
        convert(varchar, @po_no) + '</td>' +
        '<td class=inv_data>' +

```

```

        covert(varchar, @cust_no + '</td>' +
        '<td class=inv_data>' + @cust_name + '</td>' +
        '<td class=inv_data>' + @ship_via + '</td>' +
        '<td class=inv_data>' +
            covert(varchar, @ship_date, 101) + '</td>' +
        '</tr>' +
        '</table>'
EXEC sybsystemprocs.dbmail.CreateMailFile 'invoice.htm', @invoice_html,
    1, @ErrorMessage, @Ret

-- now add invoice details
SELECT invoice_html =
    '<table>' +
        '<tr><th class=inv_header>Line #</th>' +
        '<th class=inv_header>Item Code</th>' +
        '<th class=inv_header>Item Name</th>' +
        '<th class=inv_header>Quantity</th>' +
        '<th class=inv_header>Price</th>' +
    '</tr>'
EXEC sybsystemprocs.dbmail.CreateMailFile 'invoice.htm'@invoice_html,
    1, @ErrorMessage, @Ret

-- declare internal cursor to fetch ordered items
DECLARE c2 CURSOR
SELECT line_no, item, item_name, quantity, price
FROM order_items
WHERE order_no = @order_no

OPEN c2
-- get first line
FETCH c2 INTO @line_no, @item, @item_name, @quantity, @price

WHILE (@@sqlstatus = 0)
BEGIN
    SELECT invoice_html =
        '<tr><td class=inv_data>' +
            covert(varchar, line_no) + '</td>' +
        '<td class=inv_data>' + @item + '</td>' +
        '<td class=inv_data>' + @item_name + '</td>' +
        '<td class=inv_data>' +
            covert(varchar, @quantity) + '</td>' +
        '<td class=inv_data>' +
            covert(varchar, price) + '</td>' +
        '</tr>'
    EXEC sybsystemprocs.dbmail.CreateMailFile 'invoice.htm',
        @invoice_html, 1, @ErrorMessage, @Ret

    -- move to the next item
    FETCH c2 INTO @line_no, @item, @item_name, @quantity, @price
END

CLOSE c2
DEALLOCATE c2

-- now add invoice total and also save it as a file
SELECT invoice_html =
    '</table>' +
    '<hr>' +
    '<p class=totals><h3>TOTAL: ' +
        covert(varchar, @order_amount) + '</p>' +
    '</body>' +
    '</html>'
EXEC sybsystemprocs.dbmail.CreateMailFile 'invoice.htm', @invoice_html,

```



```

1, @ErrMsg, @Ret
-- fax invoice
EXEC sybsystemprocs.dbmail.SendFaxEx
    @cust_fax, 'Invoice',
    'invoice.htm', 'InvoiceCoverPage',
    @cust_name, @cust_company,
    @cust_phone, NULL,
    'This is your invoice. Call our sales department
      at (111) 222-3344 if you have any questions
      concerning this invoice',
    NULL, 'My company', 'Sales department',
    '(111) 222-3344', '(111) 222-3355',
    0, 1, 'ar@company.com', 0,
    @ErrMsg, @Ret

-- move to the next order and create/fax next invoice
FETCH c1 INTO @order_no, @order_date, @po_num, @cust_no, @cust_name,
    @ship_via, @ship_date, @order_amount, @cust_fax, @cust_phone,
    @cust_company

END

CLOSE c1
DEALLOCATE c1

```

IBM DB2

SendFax

Use this method to fax documents directly from your database or from external database-connected applications. The definition of SendFax function is shown below:

Definition

```

dbmail.SendFax(
    fax                VARCHAR(50),
    subject            VARCHAR(255),
    file_name          VARCHAR(255),
    cover_page_name    VARCHAR(255),
    recipient_name     VARCHAR(50),
    OUT SQLMessage     VARCHAR(1000),
    OUT result         INT)

```

Argument	Max Size; Value Range	Description
Fax	50	Recipient's fax number. Specify the number exactly as you would dial it from your phone number including any dial out numbers, long distance codes, local area codes and so on. Digits in the number can be separated by optional dashes, parenthesis, spaces and other non-digit symbols. For example, 1 (212) 555-6789
Subject	255	Message subject. If Cover_page_name value is not NULL and the cover page features the RE field, the subject text is used for that field. See sample cover page in Creating and modifying cover pages section for

		details.
File_Name	255	Name of the file to fax. The name must conform to standard file naming conventions and may include file path. The referenced file must exist in the directory specified by the MAILSTORE directory object.
Cover_page_name	255	Cover page file name not including file path and file extension. If NULL value is passed for the Cover_page_name argument, no cover page is attached to the fax.
Recipient_name	50	Recipient's name
SQLMessage	1000	Output variable used to return error description in case an error occurs
Result		Output variable to return procedure result

Return values:

Result returns unique message ID for the sent message (a positive number) if the procedure completes successfully.

If an error occurs, **Result** returns a negative number. Use **SQLMessage** argument to obtain the error description.

 **Usage Tips:**

- HTML and XML files used for faxing can refer to other files such as images, cascading style sheets, and so on as long as these files are accessible via the Intranet or Internet from the computer running DB Mail Server. The composite document is rendered using all referenced objects at the time the message is processed and converted to a single TIFF image.
- When developing your fax procedures you can use Internet Explorer or other web browser to preview and test the created documents. What you see in the browser Print Preview mode is what you get in the fax. Faxing HTML and XML documents is virtually the same as printing them to a printer. The only real difference is that instead of producing printed hard copies DB Mail produces TIFF images and then automatically faxes them to the destination fax number.

Example:

The following example demonstrates how to send electronic fax using DB Mail functions.

In this example we will dynamically create HTML report and then fax it.

```
BEGIN
  DECLARE ret_code INTEGER;
  DECLARE errors VARCHAR(1000);
  DECLARE report VARCHAR(4000);

  /* create dynamic HTML report */
  SET report = '<html><title>Test Report</title><body>' || chr(10) ||
    '<h2>Test Report</h2><hr>' || chr(10) ||
    '<table><tr>' || chr(10) ||
```

```

'<th bgcolor=black><font color=white>Col 1</th>' || chr(10) ||
'<th bgcolor=black><font color=white>Col 2</th></tr>' || chr(10) ||
'<tr><td>Value A</td><td align=right>1</td></tr>' || chr(10) ||
'<tr><td>Value B</td><td align=right>2</td></tr>' || chr(10) ||
'</table></body></html>';

CALL dbmail.CreateMailFile( '/maildir/report.htm', report,
                           0, errors, ret_code );

/* send report by fax */
CALL dbmail.SendFax('+1 (123) 222-3344',
                   'Test fax subject', '/maildir/report.htm'
                   'Generic', 'Test recipient',
                   errors, ret_code );

/* delete temporary report file */
CALL dbmail.DeleteMailFile( '/maildir/report.htm', errors, ret_code );

END

```

SendFaxEx

Use this method to fax documents directly from your database or from external database-connected applications. SendFaxEx is an extended version of SendFax procedure. Simply put SendFaxEx supports more options than SendFax function. The definition of SendFaxEx procedure is shown below:

Definition

```

db_mail.SendFaxEx(
    fax                VARCHAR(50),
    subject            VARCHAR(255),
    file_name          VARCHAR(255),
    cover_page_name    VARCHAR(255),
    recipient_name     VARCHAR(50),
    recipient_co       VARCHAR(50),
    recipient_phone    VARCHAR(50),
    cover_page_message VARCHAR(1000),
    sender_name        VARCHAR(50),
    sender_co          VARCHAR(50),
    sender_dept        VARCHAR(50),
    sender_phone       VARCHAR(50),
    sender_fax         VARCHAR(50),
    notify_on_success  INT,
    notify_on_failure  INT,
    notify_email       VARCHAR(100),
    priority            INT,
    OUT SQLMessage     VARCHAR(1000),
    OUT result         INT )
RETURN NUMBER

```

Argument	Max Size; Value Range	Description
Fax	50	Recipient's fax number. Specify the number exactly as you would dial it from your phone number including any dial out numbers, long distance codes, local area codes and so on. Digits in the number can be separated by optional dashes, parenthesis, spaces and other non-digit

		symbols. For example, 1 (212) 555-6789
Subject	255	Message subject. If Cover_page_name value is not NULL and the cover page features the RE field, the subject text is used for that field. See sample cover page in Creating and modifying cover pages section for details.
File_Name	255	Name of the file to fax. The name must conform to standard file naming conventions and may include file path.
Cover_page_name	255	Cover page file name not including file path and file extension. . If NULL value is passed for the Cover_page_name argument, no cover page is attached to the fax.
Recipient_name	50	Recipient's name.
Recipient_co	50	Recipients' company name, specify NULL if not available.
Recipient_phone	50	Recipient phone number. The phone number can be specified in any format and may also include phone extensions if any. For example (121) 555-3456 x123
Cover_page_message	255	Free message text that will appear in the cover page message area.
Sender_name	50	Sender's name.
Sender_co	50	Sender's company name
Sender_dept	50	Sender's department, for example <i>Sales</i> .
Sender_phone	50	Sender's phone number. The phone number can be specified in any format and may also include phone extensions if any. For example (121) 555-3456 x123
Sender_fax	50	Sender's fax number. The fax number can be specified in any format.
Notify_on_success	0..1	Whether to notify message sender by email after fax transmission completed successfully. If this value is 1, in case of a successful transmission DB Mail will email notification to the email address specified in Notify_Email argument.
Notify_on_failure	0..1	Whether to notify message sender by email after fax transmission failed. If this value is 1, in case of a failed transmission DB Mail will email notification to the email address specified in Notify_Email argument.
Notify_Email	100	Email address of the person or email group that will receive email notification in case Notify_on_success or Notify_on_failure value is 1.
SQLMessage	1000	Output variable used to return error description in case an error occurs
Result		Output variable to return procedure result

Return values:

Result returns unique message ID for the sent message (a positive number) if the procedure

completes successfully.

If an error occurs, **Result** returns a negative number. Use **SQLMessage** argument to obtain the error description.

Usage Tips:

- HTML and XML files used for faxing can refer to other files such as images, cascading style sheets, and so on as long as these files are accessible via the Intranet or Internet from the computer running DB Mail Server. The composite document is rendered using all referenced objects at the time the message is processed and converted to a single TIFF image.
- When developing your fax procedures you can use Internet Explorer or other web browser to preview and test the created documents. What you see in the browser Print Preview mode is what you get in the fax. Faxing HTML and XML documents is virtually the same as printing them to a printer. The only real difference is that instead of producing printed hard copies DB Mail produces TIFF images and then automatically faxes them to the destination fax number.

Example:

In this example we will dynamically create invoices for all orders placed yesterday and then fax them to customers.

```
BEGIN
  DECLARE ret_code INTEGER;
  DECLARE errors VARCHAR(1000);
  DECLARE invoice_html VARCHAR(4000);
  DECLARE order_no_dup INTEGER;

  FOR each_record AS cursor1 CURSOR FOR
    SELECT order_no, order_date, po_num, cust_no,
           cust_name, ship_via, ship_date, order_amount,
           cust_fax, cust_phone, cust_company
    FROM orders
    WHERE order_date = CURRENT DATE - DAY(1)
DO

  -- make invoice header
  SET invoice_html =
    '<html><head>' ||
    '  <title>Invoice #' || char(order_no) || '</title>' ||
    '  <link rel=stylesheet href="http://www.company.com/style.css">' ||
    '</head>' ||
    '<body>' ||
    '  <h2>Invoice #' || char(order_no) || '</h2><hr>' ||
    '  <table>' ||
    '    <tr><th class=inv_header>Order Date</th>' ||
    '      <th class=inv_header>PO #</th>' ||
    '      <th class=inv_header>Customer #</th>' ||
    '      <th class=inv_header>Customer Name</th>' ||
    '      <th class=inv_header>Ship Via</th>' ||
    '      <th class=inv_header>Ship Date</th>' ||
    '</tr>' ||
    '  <tr><td class=inv_data>' || char(order_date) || '</td>' ||
    '    <td class=inv_data>' || po_no || '</td>' ||
    '    <td class=inv_data>' || char(cust_no) || '</td>' ||
    '    <td class=inv_data>' || cust_name || '</td>' ||
    '    <td class=inv_data>' || ship_via || '</td>' ||
```

```

        '<td class=inv_data>' || char(ship_date) || '</td>' ||
    '</tr>' ||
    '</table>';

CALL dbmail.CreateMailFile('invoice.htm', invoice_html,
    0, errors, ret_code );

-- now add invoice details
SET invoice_html =
    '<table>' ||
        '<tr><th class=inv_header>Line #</th>' ||
        '<th class=inv_header>Item Code</th>' ||
        '<th class=inv_header>Item Name</th>' ||
        '<th class=inv_header>Quantity</th>' ||
        '<th class=inv_header>Price</th>' ||
    '</tr>';

CALL dbmail.CreateMailFile('invoice.htm', invoice_html,
    1, errors, ret_code );

SET order_no_dup = order_no;

FOR each_record AS cursor2 CURSOR FOR
    SELECT line_no, item, item_name, quantity, price
    FROM order_items
    WHERE order_no = order_no_dup
DO
    SET invoice_html =
        '<tr><td class=inv_data>' || char(line_no) || '</td>' ||
        '<td class=inv_data>' || item || '</td>' ||
        '<td class=inv_data>' || item_name || '</td>' ||
        '<td class=inv_data>' || char(quantity) || '</td>' ||
        '<td class=inv_data>' || char(price) || '</td>' ||
    '</tr>';

    CALL dbmail.CreateMailFile('invoice.htm', invoice_html,
        1, errors, ret_code );

    -- move to the next item
END FOR;

-- now add invoice total and save it as a file
SET invoice_html = '</table>' ||
    '<hr>' ||
    '<p class=totals><h3>TOTAL: ' || rec.order_amount || '</p>' ||
    '</body>' ||
    '</html>';

CALL dbmail.CreateMailFile('invoice.htm', invoice_html,
    1, errors, ret_code );

-- fax invoice
CALL dbmail.SendFaxEx(
    cust_fax,
    'Invoice #' || to_char(order_no),
    'invoice.htm',
    'InvoiceCoverPage',
    cust_name,
    cust_company,
    cust_phone,
    NULL,
    'This is your invoice. Call our sales department ' ||
    'at (111) 222-3344 if you have any questions ' ||

```

```
        'concerning this invoice',
        NULL,
        'My company',
        'Sales department',
        '(111) 222-3344', '(111) 222-3355',
        0, 1,
        'ar@company.com', 0,
        errors, ret_code );

    -- move to the next order and create/fax next invoice
END FOR;
END
```

CHAPTER 11, Sending phone/voice messages

Overview

To make phone calls and send automated voice messages from database applications use the *dbmail.SendVoice* method. This method can be used to send pre-recorded messages in standard WAV file format, send dynamically synthesized voice messages using text-to-speech technology and send messages that contain multiple parts including pre-recorded and dynamic segment. DB Mail communicates to the VoMS server which actually makes the required phone calls, verifies message format and files, assembles multiple message parts into a single message, performs call progress monitoring, detects call answer and then "speaks" the resulting message.

Advanced options

VoMS server supports a number of options that can be used to customize voice message delivery. The following options can be configured using VoMS Administrator Console:

- Modem selection and configuration
- Default computer voice for dynamically synthesized voice messages, including voice name, rate and volume
- Call progress detection parameters, timeouts, number of retries.
- VoMS server security and users
- Other message queuing and processing options

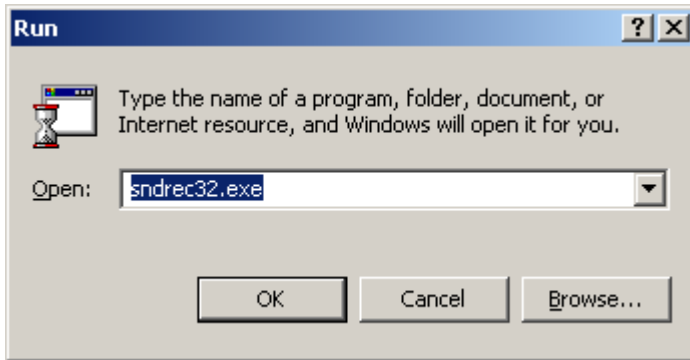
For more information on VoMS software usage and configuration see VoMS manual.

For detailed descriptions of the *dbmail.SendVoice* method and usage examples specific to your database system refer to the following topics in this chapter.

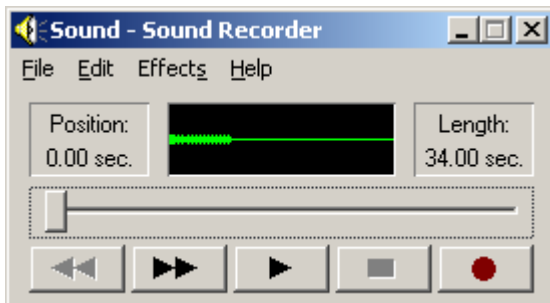
Creating pre-recorded sound messages and message segments

Use standard Windows Sound Recorder utility or similar software to create pre-recorded messages and save them as .WAV file. To record sounds you must have a microphone and sound card installed on your computer.

A shortcut to the standard Windows Sound Recorder utility can be normally found in **C:\Documents and Settings\All Users\Start Menu\Programs\Accessories\Entertainment** folder. The utility can be also started using **Run** option in the Windows Start menu. The name of this program is sndrec32.exe.



The following screenshot demonstrates the Sound Recorder utility graphical interface.



For detailed instruction on how to use this utility click Help/Contents menu available in the Sound Recorder graphical interface. After you are done with the sound recording copy all created .WAV files to the directory accessible from your database server. If your database server is running on a Unix or other non-Windows system you can use FTP or other appropriate methods to copy WAV files.

The recorded WAV files can be then attached to voice messages using *dbmail.AttachFile* method just like any other files can be attached to email message sent using *dmail.SendMail* method.

For detailed descriptions of how to send sound files and specify their position in multiple-part messages see description of the *dbmail.SendVoice* method and usage examples specific to your database system in the following topics in this chapter.

Creating dynamically synthesized voice messages and message segments

VoMS software features built-in methods for dynamically synthesized voice messages using text-to-speech technology provided with most Windows installations. DB Mail utilizes these methods when converting regular text messages and multiple-part messages containing text segments to WAV compatible sound files. On your part, you simply call *dbmail.SendVoice* method and specify text that you want to the computer to speak to the message recipient using computer synthesized human voice. DB Mail and VoMs software automatically take care of the rest. They automatically convert text messages to sound files, dial specified phone numbers, analyze call recipient responses are speak your messages in case if a successfully human voice or answering machine response is detected.

Oracle


SEND_VOICE

Use this method to make phone calls and play sound messages directly from your database or from external database-connected applications. The definition of the SEND_VOICE function is shown below:

Definition

```
db_mail.send_voice(
    telephones      VARCHAR2,
    message         VARCHAR2,
    priority        INTEGER DEFAULT 0,
    attachment_id   NUMBER DEFAULT NULL)
RETURN NUMBER
```

Argument	Max Size; Value Range	Description
Telephones	4000	<p>Recipient's phone numbers. Specify the number exactly as you would dial it from your phone number including any dial out numbers, long distance codes, local area codes and so on. Digits in the number can be separated by optional dashes, parenthesis, spaces and other non-digit symbols. For example, 1 (212) 555-6789</p> <p>If you need to send the same message to multiple recipients, use comma to separate multiple phone numbers.</p>
Message	4000	<p>Message description containing optional text you want the computer to speak to the phone call recipient and also containing optional pre-recorded sound files. Use the Attachment_ID argument to specify which pre-recorded sound files or BLOB values containing sound data you want to use with the message. To specify message file position within the message text use a pair of <wav> and </wav> tags. For example,</p> <p><i>Say <wav>Hello.wav</wav> to my friend</i></p> <p>This message text will instruct DB Mail to perform the following processing steps:</p> <ol style="list-style-type: none"> 1. Convert word "Say" and the trailing space character into a WAV file using computer synthesized human voice. 2. Append pre-recorded hello.wav file to the WAV file created in step 1. 3. Convert words "to my friend" and the leading space character into a WAV file using computer synthesized human voice. 4. Append last WAV file created in step 3 to the file created in step 1 and updated in step 2. <p>Here is another example that uses 2 pre-recorded message segments and text segment as a computer synthesized text-to-speech insertion.</p> <p><i><wav>appointment_reminder</wav> 10:00 AM</i></p>

		<p><wav>call_to_cancel.wav</wav></p> <p>This message text will instruct DB Mail to perform the following processing steps:</p> <ol style="list-style-type: none"> 1. Convert text "10:00 AM " including leading and trailing spaces into a WAV file using computer synthesized human voice. 2. Concatenate <i>appointment_reminder.wav</i> file with the WAV created in step 2 and then with <i>call_to_cancel.wav</i> file. <p> Tip: If you need to send a message whose text is longer than the maximum size of varchar data-type in your database you can use a special attachment file with the name <i>message</i> and no extension. DB Mail will use the contents of that file for the message text. The value of the Message parameter will be ignored in that case. For more information see Error! Not a valid bookmark self-reference. topic.</p>
Priority (optional)	0..2	<p>Message processing priority takes a value of 0, 1, or 2. Note that this property is used in 2 different places. If Message Queuing is enabled in DB Mail options, messages having higher priority numbers are processed before messages having lower priority numbers. VoMS server software maintains its own message queue. Message priorities have similar effect on the VoMS server message processing, messages having higher priority numbers are processed before messages having lower priority numbers.</p>
Attachment_ID (optional)		<p>ID of the record or group of records in the MAIL_ATTACH table containing attachment data. Message attachment should be created using <i>DB_MAIL.ATTACH_FILE</i> and <i>DB_MAIL.ATTACH_DATA</i> methods.</p> <p>Attached files and data must be in the standard WAV file format.</p>

Return values:

0 - Success.

1 - Timed out. This procedure can timeout either because it cannot get a lock on the pipe, or because the pipe remains too full to be used.

3 - An interrupt occurred.

 **Usage Tips:**

- One message can have any number of text and sound segments with any positions within the message.
- You may not use nested **<wav>**tags. Nested tags will lead to message processing errors.
- Spaces in the message text have special importance. They are translated into pauses between words and message segments.
- To send a message containing only pre-recorded sound file use simple **<wav>filename.wav</wav>** message text, where *filename.wav* should be replaced with the actual file name.

- When creating pre-recorded sound messages make sure to set sound volume in all files to the same level.

Examples:

The following examples demonstrate how to make phone calls and send voice messages using DB Mail functions.

Example 1 (SQL):

The following SELECT statement will call all people who have an appointment scheduled between 9:00 AM and 9:00 PM and remind them about appointment time. In this example we will dynamically synthesize human voice to speak the entire message. This example also assumes that the APPOINTMENTS table contains the APP_TIME, DOCTOR_NAME, PATIENT_NAME, and PHONE columns.

```
SELECT db_mail.send_voice(PHONE,
    'This is a reminder that ' || PATIENT_NAME ||
    ' has an appointment with doctor ' || DOCTOR_NAME ||
    ' today at ' || to_char(APP_TIME, 'HH:MI AP') ||
    ' If you are unable to come please call 800-123-4567 ' ||
    ' to cancel this appointment.')
FROM APPOINTMENTS
WHERE APP_TIME BETWEEN trunc(sysdate) + 9/24 AND trunc(sysdate) + 21/24;
```

Example 2 (Oracle 8, 8i, 9i, and 10g; 1 message with 1 pre-recorded segment stored as an external WAV file):

This is a more advanced PL/SQL example that demonstrates how to use DB_MAIL.SEND_VOICE and DB_MAIL.ATTACH_FILE functions to send pre-recorded voice message. The pre-recorded segment is stored as WAV file in the directory referenced by SOUND_FILES directory object.

```
DECLARE
    attach_id INTEGER;
    ret_code INTEGER;
BEGIN
    attach_id := db_mail.Attach_File(NULL, 'announce.wav', 'SOUND_FILES');

    ret_code := db_mail.Send_Voice('+1 (123) 456-7890',
        '<wav>announce.wav</wav>',
        0,
        attach_id);
END;
```

Example 3 (Oracle 8, 8i, 9i, and 10g; 1 message with 3 pre-recorded segments stored as external WAV files):

This example demonstrates how to use DB_MAIL.SEND_VOICE and DB_MAIL.ATTACH_FILE functions to send pre-recorded voice message. The message contains 3 pre-recorded segments which are stored as WAV files in the directory referenced by SOUND_FILES directory object.

```
DECLARE
    attach_id INTEGER;
    ret_code INTEGER;
```

```

BEGIN
  attach_id := db_mail.Attach_File(NULL, 'greeting.wav', 'SOUND_FILES');
  ret_code := db_mail.Attach_File(attach_id, 'announce.wav', 'SOUND_FILES');
  ret_code := db_mail.Attach_File(attach_id, 'bye.wav', 'SOUND_FILES');

  ret_code := db_mail.Send_Voice('+1 (123) 456-7890',
    '<wav>greeting.wav</wav> <wav>announce.wav</wav> <wav>bye.wav</wav>',
    0,
    attach_id);
END;

```

Example 4 (Oracle 8, 8i, 9i, and 10g; 1 message with 1 pre-recorded segment as an internal BLOB data):

This example demonstrates how to use DB_MAIL.SEND_VOICE and DB_MAIL.ATTACH_DATA functions to send pre-recorded voice message. The pre-recorded segment is stored as WAV data in a database table BLOB type column.

```

DECLARE
  attach_id INTEGER;
  ret_code INTEGER;
  wav_data BLOB;
BEGIN
  SELECT b_col
  INTO wav_data
  FROM lob_table
  WHERE key_value = 21;

  attach_id := db_mail.Attach_Data(NULL, 1, 'greeting.wav', wav_data);

  ret_code := db_mail.Send_Voice('+1 (123) 456-7890',
    'Attention! <wav>greeting.wav</wav> Bye!',
    0,
    attach_id);
END;

```

Example 5 (Oracle 8, 8i, 9i, and 10g; 1 message with 3 pre-recorded segments as an internal BLOB data):

This example demonstrates how to use DB_MAIL.SEND_VOICE and DB_MAIL.ATTACH_DATA functions to send pre-recorded voice message. The message consists of 3 pre-recorded segments which are stored as WAV data in a database table BLOB type column.

```

DECLARE
  attach_id INTEGER;
  ret_code INTEGER;
  wav_data BLOB;
BEGIN
  SELECT b_col
  INTO wav_data
  FROM lob_table
  WHERE file_name = 'greeting.wav';

  attach_id := db_mail.Attach_Data(NULL, 1, 'greeting.wav', wav_data);

  SELECT b_col
  INTO wav_data
  FROM lob_table

```

```

WHERE file_name = 'announce.wav';

ret_code := db_mail.Attach_Data(attach_id, 2, 'announce.wav', wav_data);

SELECT b_col
INTO wav_data
FROM lob_table
WHERE file_name = 'bye.gif';

ret_code := db_mail.Attach_Data(attach_id, 3, 'bye.wav', wav_data);

ret_code := db_mail.Send_Voice('+1 (123) 456-7890',
    '<wav>greeting.wav</wav> <wav>announce.wav</wav> <wav>bye.wav</wav>',
    0,
    attach_id);
END;
```

Example 6 (Oracle 8i, 9i and 10g; many messages sharing the same voice segment stored as an external WAV file):

This example demonstrates how to use DB_MAIL.SEND_VOICE and DB_MAIL.ATTACH_FILE functions to send pre-recorded voice message. The pre-recorded segment is stored as WAV file in the directory referenced by SOUND_FILES directory object.

```

SELECT db_mail.send_voice(PATIENT_PHONE,
    '<wav>reminder.wav</wav>'
    0,
    db_mail.Attach_File(NULL, 'reminder.wav', 'SOUND_FILES')) ;
FROM APPOINTMENTS
WHERE APP_TIME BETWEEN trunc(sysdate) + 9/24 AND trunc(sysdate) + 21/24;
```

Example 7 (Oracle 8i, 9i and 10g, many messages sharing 2 voice segments stored as external WAV files):

This example demonstrates how to use DB_MAIL.SEND_VOICE and DB_MAIL.ATTACH_FILE functions to send pre-recorded voice message. The example message contains 2 pre-recorded segments stored as WAV files in the directory referenced by SOUND_FILES directory object.

```

SELECT db_mail.send_voice(PATIENT_PHONE,
    '<wav>reminder.wav</wav>'
    0,
    db_mail.Attach_File(Attach_File(NULL,
        'hello.wav', 'SOUND_FILES'),
        'reminder.wav', 'SOUND_FILES')) ;
FROM APPOINTMENTS
WHERE APP_TIME BETWEEN trunc(sysdate) + 9/24 AND trunc(sysdate) + 21/24;
```

Microsoft SQL Server


SendVoice

Use this method to make phone calls and play sound messages directly from your database or from

external database-connected applications. The definition of the SEND_VOICE function is shown below:

Definition

```
dbmail.SendVoice(
  @telephones    VARCHAR(8000),
  @message       VARCHAR(8000),
  @priority      INT = 0,
  @attachment_id INT = NULL)
```

Argument	Max Size; Value Range	Description
Telephones	8000	<p>Recipient's phone numbers. Specify the number exactly as you would dial it from your phone number including any dial out numbers, long distance codes, local area codes and so on. Digits in the number can be separated by optional dashes, parenthesis, spaces and other non-digit symbols. For example, 1 (212) 555-6789</p> <p>If you need to send the same message to multiple recipients, use comma to separate multiple phone numbers.</p>
Message	8000	<p>Message description containing optional text you want the computer to speak to the phone call recipient and also containing optional pre-recorded sound files. Use the Attachment_ID argument to specify which pre-recorded sound files or BLOB values containing sound data you want to use with the message. To specify message file position within the message text use a pair of <wav> and </wav> tags. For example,</p> <p><i>Say <wav>Hello.wav</wav> to my friend</i></p> <p>This message text will instruct DB Mail to perform the following processing steps:</p> <ol style="list-style-type: none"> 1. Convert word "Say" and the trailing space character into a WAV file using computer synthesized human voice. 2. Append pre-recorded hello.wav file to the WAV file created in step 1. 3. Convert words "to my friend" and the leading space character into a WAV file using computer synthesized human voice. 4. Append last WAV file created in step 3 to the file created in step 1 and updated in step 2. <p>Here is another example that uses 2 pre-recorded message segments and text segment as a computer synthesized text-to-speech insertion.</p> <p><i><wav>appointment_reminder</wav> 10:00 AM <wav>call_to_cancel.wav</wav></i></p> <p>This message text will instruct DB Mail to perform the following processing steps:</p> <ol style="list-style-type: none"> 1. Convert text "10:00 AM" including leading and trailing spaces into a WAV file using computer synthesized human voice. 2. Concatenate <i>appointment_reminder.wav</i> file with the WAV created in step 2 and then with <i>call_to_cancel.wav</i> file. <p> Tip: If you need to send a message whose text is longer than the maximum size of varchar data-type in your database you can use a special attachment file with the name <i>message</i> and no extension. DB Mail will use the contents of that file for the message text. The value of</p>

		the Message parameter will be ignored in that case. For more information see Error! Not a valid bookmark self-reference. topic.
Priority (optional)	0..2	Message processing priority takes a value of 0 , 1 , or 2 . Note that this property is used in 2 different places. If Message Queuing is enabled in DB Mail options, messages having higher priority numbers are processed before messages having lower priority numbers. VoMS server software maintains its own message queue. Message priorities have similar effect on the VoMS server message processing, messages having higher priority numbers are processed before messages having lower priority numbers.
Attachment_ID (optional)		ID of the record or group of records in the MAIL_ATTACH table containing attachment data. Message attachment should be created using <i>dbmail.AttachFile</i> and <i>dbmail.AttachData</i> methods. Attached files and data must be in the standard WAV file format.

Return values: Returns unique message ID or returns -1 if an error occurs.



Usage Tips:

- One message can have any number of text and sound segments with any positions within the message.
- You may not use nested **<wav>** tags. Nested tags will lead to message processing errors.
- Spaces in the message text have special importance. They are translated into pauses between words and message segments.
- To send a message containing only pre-recorded sound file use simple **<wav>filename.wav</wav>** message text, where *filename.wav* should be replaced with the actual file name.
- When creating pre-recorded sound messages make sure to set sound volume in all files to the same level.
- Use the @@ERROR global variable to check for errors. @@ERROR is set to 0 if the procedure executed successfully. If an error occurs, a non-zero number is returned. @@ERROR returns the number of the error message until another Transact-SQL statement is executed. You can view the text associated with an @@ERROR error number in the SYSMESSAGES system table.

Examples

The following examples demonstrate how to make phone calls and send voice messages using DB Mail procedures.

Example 1 (Sending simple dynamically synthesized message to 1 recipient):

The following EXECUTE statement will send "Password expiring" reminder message to telephone number +1 (123) 345-6789. In this example we will dynamically synthesize human voice to speak the entire message.

```
EXEC master.dbmail.SendVoice '+1 (123) 345-6789',
```



```
'Attention batch job owners, your database password will
  expire in 3 days. Be sure to update your batch jobs before
  that date. If you need assistance, reply to this message
  with your questions'
```

Example 2 (Sending dynamically synthesized messages to multiple recipients):

In this example we create user defined scalar function callable from various SQL statement just like other built-in system functions. Using this function we will send email to all users who run batch jobs and whose database passwords are going to expire next Monday. In this example we will dynamically synthesize human voice to speak the entire message. This example also assumes that the BATCH_USER table contains the EXPIRE_DATE and FNAME columns, which represent the expiry date of the password, the email user id and the full user name.

```
-- First, let's create a user-defined scalar Transact-SQL function that
-- we can call from SELECT statements
CREATE FUNCTION mySendVoice( @telephone VARCHAR(30),
                             @message VARCHAR(8000) )
RETURNS INT
AS
BEGIN
    DECLARE @ret INT
    EXEC @ret = master.dbmail.SendVoice @ telephone, @message
    RETURN (@ret)
END
go

-- Now, we can call our own send voice function
SELECT mySendVoice(userid + '@domain.com',
  'Dear ' + fname + ', ' + char(10) + char(10) +
  'Your database password will expire next Monday. ' +
  'Be sure to update your batch jobs before that date.' + char(10) +
  'If you need assistance, call our help desk (800) 555-6677' )
FROM batch_user
WHERE expire_date =
    DateAdd(DAY, 3, convert(datetime, convert(varchar, GetDate(), 101)))
go
```

Example 3 (Sending dynamically synthesized message from a T-SQL procedure):

This is a more advanced T-SQL example of a similar use of the dbmail.SendVoice procedure:

```
/******
* This procedure debits the specified account by specified amount
* only if there are sufficient funds to cover the withdrawal, and if there
* are not, it sends an automated phone call to the account holder
* notifying about insufficient funds problem.
* This function returns new account balance.
*****/
CREATE PROCEDURE sp_acct_debit (@acct_nbr CHAR(10), @debit_amt MONEY)
AS
BEGIN
    DECLARE
        @acct_balance MONEY,
        @ret_code INTEGER
    SET @ret_code = -1

    SELECT @acct_balance = balance
    FROM accounts
```

```

WHERE acct = @acct_nbr

IF @acct_balance >= @debit_amt
BEGIN
    SET @acct_balance = @acct_balance - @debit_amt
    UPDATE accounts
    SET balance = @acct_balance
    WHERE acct = @acct_nbr

    IF @@error = 0 SET @ret_code = 1 -- success
END
ELSE
    -- Insufficient funds.
    -- Send asynchronous voice notification to the account holder
    DECLARE @phone VARCHAR(50),
            @message VARCHAR(200)

    SELECT @phone = phone,
           @message = 'Best Bank could not complete your last ' +
                      'debit transaction for account number ending with ' +
                      substr(@acct_nbr, 7, 4) + ' because there are ' +
                      'insufficient funds available in your account.'
    FROM accounts
    WHERE acct = @acct_nbr

    EXEC @rec_code = master.dbmail.SendVoice @phone, @message

    SET @ret_code = 0
END

RETURN (@ret_code) -- return code 1 indicates success
                -- 0 indicates insufficient funds
                -- -1 indicates all other problems

END

go

```

Example 4 (Sending 1 message to 1 recipient using pre-recorded sound message stored as external file):

```

DECLARE @attach_id INTEGER

EXEC @attach_id = master.dbmail.AttachFile NULL, 'c:\announcement\new.wav'

EXEC master.dbmail.SendVoice '+1 (123) 345-6789',
    '<wav>new.wav</wav>',
    0,
    @attach_id

```

Example 5 (Sending multiple-part message containing 1 dynamically synthesized segment and 2 pre-recorded sound segments stored as external files):

```

DECLARE @attach_id INTEGER

EXEC @attach_id = master.dbmail.AttachFile NULL, 'c:\misc\dear.wav'
EXEC master.dbmail.AttachFile @attach_id, 'c:\campaigns\donate.wav'

EXEC master.dbmail.SendVoice '+1 (123) 345-6789',
    '<wav>dear.wav</wav> John ' <wav>donate.wav</wav>',
    0,

```

```
@attach_id
```

Example 6 (Sending 1 message to 1 recipient using pre-recorded sound message stored as internal BLOB in an image data-type column):

```
DECLARE @attach_id INTEGER
DECLARE @data_length INT
DECLARE @ptrval BINARY(16)

-- read blob value from a table
SELECT @data_length = DATALENGTH(blob_column),
       @ptrval = TEXTPTR(blob_column)
FROM sound_table
WHERE sound_id = 21

READTEXT sound_table.blob_column @ptrval 1 @data_length

-- pass that value to the AttachData procedure
EXECUTE @attach_id = master.dbmail.AttachData NULL, 'product.gif', @ptrval

-- now, dial out and speak the message
EXEC master.dbmail.SendVoice '+1 (123) 345-6789',
    '<wav>new.wav</wav>',
    0,
    @attach_id
```

Example 7 (Sending 1 message to multiple recipients using 2 pre-recorded sound segments stored as internal BLOB values in an image data-type column and 1 dynamically synthesized segment whose text is based on a table data):

```
-- First, let's create a user-defined scalar Transact-SQL function that
-- we can call from SELECT statements
CREATE FUNCTION mySendVoice( @phone VARCHAR(30),
                           @variable_part VARCHAR(255) )
RETURNS INT
AS
BEGIN
    DECLARE @attach_id INTEGER
    DECLARE @data_length INT
    DECLARE @ptrval BINARY(16)

    -- read first blob value from a table
    SELECT @data_length = DATALENGTH(blob_column),
           @ptrval = TEXTPTR(blob_column)
    FROM sound_table
    WHERE sound_id = 1

    READTEXT sound_table.blob_column @ptrval 1 @data_length
    -- pass that value to the AttachData procedure
    EXEC @attach_id = master.dbmail.AttachData NULL, 'new.wav', @ptrval

    -- read second blob value from a table
    SELECT @data_length = DATALENGTH(blob_column),
           @ptrval = TEXTPTR(blob_column)
    FROM sound_table
    WHERE sound_id = 2

    READTEXT sound_table.blob_column @ptrval 1 @data_length
    -- pass that value to the AttachData procedure
    EXEC master.dbmail.AttachData @attach_id, 'announce.wav', @ptrval
```

```

-- now, dial out and speak the entire message
EXEC @ret = EXEC master.dbmail.SendVoice @phone,
      '<wav>new.wav</wav>' + @variable_part + '<wav>announce.wav</wav>',
      0,
      @attach_id

RETURN (@ret)
END
go

-- Now, we can call our own send voice function
SELECT mySendVoice(customer_phone,
                  'Dear ' + customer_title + ' ' + customer_name)
FROM customer
WHERE customer_phone IS NOT NULL
      AND status = 'A';
go

```

Sybase SQL Server, ASE, ASA

SendVoice

Use this method to make phone calls and play sound messages directly from your database or from external database-connected applications. The definition of the SEND_VOICE function is shown below:


Definition

```

dbmail.SendVoice(
    @telephones    VARCHAR(255),
    @message       VARCHAR(255),
    @priority      INT = 0,
    @attachment_id INT = NULL,
    OUT SQLMessage VARCHAR(255),
    OUT result     INT)

```

Argument	Max Size; Value Range	Description
Telephones	255	Recipient's phone numbers. Specify the number exactly as you would dial it from your phone number including any dial out numbers, long distance codes, local area codes and so on. Digits in the number can be separated by optional dashes, parenthesis, spaces and other non-digit symbols. For example, 1 (212) 555-6789 If you need to send the same message to multiple recipients, use comma to separate multiple phone numbers.
Message	255	Message description containing optional text you want the computer to speak to the phone call recipient and also containing optional pre-recorded sound files. Use the Attachment_ID argument to specify which pre-recorded sound files or BLOB values containing sound data you want to use with the message. To specify message file position within the message text use a pair of <wav> and </wav> tags. For example,

		<p><i>Say <wav>Hello.wav</wav> to my friend</i></p> <p>This message text will instruct DB Mail to perform the following processing steps:</p> <ol style="list-style-type: none"> 1. Convert word "Say" and the trailing space character into a WAV file using computer synthesized human voice. 2. Append pre-recorded hello.wav file to the WAV file created in step 1. 3. Convert words "to my friend" and the leading space character into a WAV file using computer synthesized human voice. 4. Append last WAV file created in step 3 to the file created in step 1 and updated in step 2. <p>Here is another example that uses 2 pre-recorded message segments and text segment as a computer synthesized text-to-speech insertion.</p> <p><i><wav>appointment_reminder</wav> 10:00 AM <wav>call_to_cancel.wav</wav></i></p> <p>This message text will instruct DB Mail to perform the following processing steps:</p> <ol style="list-style-type: none"> 1. Convert text "10:00 AM " including leading and trailing spaces into a WAV file using computer synthesized human voice. 2. Concatenate <i>appointment_reminder.wav</i> file with the WAV created in step 2 and then with <i>call_to_cancel.wav</i> file. <p> Tip: If you need to send a message whose text is longer than the maximum size of varchar data-type in your database you can use a special attachment file with the name <i>message</i> and no extension. DB Mail will use the contents of that file for the message text. The value of the Message parameter will be ignored in that case. For more information see Error! Not a valid bookmark self-reference. topic.</p>
Priority (optional)	0..2	Message processing priority takes a value of 0 , 1 , or 2 . Note that this property is used in 2 different places. If Message Queuing is enabled in DB Mail options, messages having higher priority numbers are processed before messages having lower priority numbers. VoMS server software maintains its own message queue. Message priorities have similar effect on the VoMS server message processing, messages having higher priority numbers are processed before messages having lower priority numbers.
Attachment_ID (optional)		ID of the record or group of records in the MAIL_ATTACH table containing attachment data. Message attachment should be created using <i>dbmail.AttachFile</i> and <i>dbmail.AttachData</i> methods. Attached files and data must be in the standard WAV file format.
SQLMessage	255	Output variable used to return error description in case an error occurs
Result		Output variable to return procedure result

Return values: Returns unique message ID or returns -1 if an error occurs.



Usage Tips:

- One message can have any number of text and sound segments with any positions within

the message.

- You may not use nested `<wav>` tags. Nested tags will lead to message processing errors.
- Spaces in the message text have special importance. They are translated into pauses between words and message segments.
- To send a message containing only pre-recorded sound file use simple `<wav>filename.wav</wav>` message text, where `filename.wav` should be replaced with the actual file name.
- When creating pre-recorded sound messages make sure to set sound volume in all files to the same level.
- Use the `@@ERROR` global variable to check for errors. `@@ERROR` is set to 0 if the procedure executed successfully. If an error occurs, a non-zero number is returned. `@@ERROR` returns the number of the error message until another Transact-SQL statement is executed. You can view the text associated with an `@@ERROR` error number in the SYSMESSAGES system table.

Examples

The following examples demonstrate how to make phone calls and send voice messages using DB Mail procedures.

Example 1 (Sending simple dynamically synthesized message to 1 recipient):

The following EXECUTE statement will send "Password expiring" reminder message to telephone number +1 (123) 345-6789. In this example we will dynamically synthesize human voice to speak the entire message.

```
DECLARE @Ret INT, @ErrMsg VARCHAR(255)

EXEC sysystemprocs.dbmail.SendVoice '+1 (123) 345-6789',
    'Attention batch job owners, your database password will
     expire in 3 days. Be sure to update your batch jobs before
     that date. If you need assistance, reply to this message
     with your questions', @ErrMsg, @Ret
```

Example 2 (Sending dynamically synthesized message from a T-SQL procedure):

This is a more advanced T-SQL example of a similar use of the dbmail.SendVoice procedure:

```
/******
 * This procedure debits the specified account by specified amount
 * only if there are sufficient funds to cover the withdrawal, and if there
 * are not, it sends an automated phone call to the account holder
 * notifying about insufficient funds problem.
 * This function returns new account balance.
 *****/
CREATE PROCEDURE sp_acct_debit (@acct_nbr CHAR(10), @debit_amt MONEY)
AS
BEGIN
    DECLARE
        @acct_balance MONEY,
        @ret_code INTEGER,
```

```

DECLARE
    @Ret INT,
    @ErrorMessage VARCHAR(255)

SET @ret_code = -1

SELECT @acct_balance = balance
FROM accounts
WHERE acct = @acct_nbr

IF @acct_balance >= @debit_amt
BEGIN
    SET @acct_balance = @acct_balance - @debit_amt
    UPDATE accounts
    SET balance = @acct_balance
    WHERE acct = @acct_nbr

    IF @@error = 0 SET @ret_code = 1 -- success
END
ELSE
    -- Insufficient funds.
    -- Send asynchronous voice notification to the account holder
    DECLARE @phone VARCHAR(50),
            @message VARCHAR(200)

    SELECT @phone = phone,
           @message = 'Best Bank could not complete your last ' +
                    'debit transaction for account number ending with ' +
                    substr(@acct_nbr, 7, 4) + ' because there are ' +
                    'insufficient funds available in your account.'
    FROM accounts
    WHERE acct = @acct_nbr

    EXEC @rec_code = syssystemprocs.dbmail.SendVoice @phone, @message,
                                                    @ErrorMessage, @Ret

    SET @ret_code = 0
END

RETURN (@ret_code) -- return code 1 indicates success
                -- 0 indicates insufficient funds
                -- -1 indicates all other problems

END

go

```

Example 3 (Sending 1 message to 1 recipient using pre-recorded sound message stored as external file):

```

DECLARE @attach_id INTEGER
DECLARE @Ret INT, @ErrorMessage VARCHAR(255)

EXEC @attach_id = syssystemprocs.dbmail.AttachFile NULL,
                'c:\announcement\new.wav',
                @ErrorMessage, @Err

EXEC syssystemprocs.dbmail.SendVoice '+1 (123) 345-6789',
    '<wav>new.wav</wav>', 0, @attach_id, @ErrorMessage, @Err

```

Example 4 (Sending 1 message to 1 recipient using pre-recorded sound message stored as internal BLOB in an image data-type column):

```

DECLARE @attach_id INTEGER
DECLARE @ptrval BINARY(16)
DECLARE @Ret INT, @ErrorMessage VARCHAR(255)

-- read blob value from a table
SELECT @ptrval = TEXTPTR(sound_col)
FROM sound_table
WHERE sound_id = 21

-- pass that value to the AttachData procedure
EXECUTE @attach_id = sybssystemprocs.dbmail.AttachData NULL,
          'announce.wav', @ptrval,
          @ErrorMessage, @Err

-- now, dial out and speak the message
EXEC sybssystemprocs.dbmail.SendVoice '+1 (123) 345-6789',
    '<wav>announce.wav</wav>', 0, @attach_id, @ErrorMessage, @Err

```

Example 5 (Sending multiple-part message containing 1 dynamically synthesized segment and 2 pre-recorded sound segments stored as external files):

```

DECLARE @attach_id INTEGER
DECLARE @Ret INT, @ErrorMessage VARCHAR(255)

EXEC @attach_id = sybssystemprocs.dbmail.AttachFile NULL, 'c:\misc\dear.wav',
    @ErrorMessage, @Err
EXEC sybssystemprocs.dbmail.AttachFile @attach_id, 'c:\campaigns\donate.wav',
    @ErrorMessage, @Err

EXEC sybssystemprocs.dbmail.SendVoice '+1 (123) 345-6789',
    '<wav>dear.wav</wav> John ' <wav>donate.wav</wav>',
    0, @attach_id, @ErrorMessage, @Ret

```

IBM DB2

SendVoice


Use this method to make phone calls and play sound messages directly from your database or from external database-connected applications. The definition of the SEND_VOICE function is shown below:

Definition

```

dbmail.SendVoice(
    telephones      VARCHAR( 32672 ),
    message         VARCHAR( 32672 ),
    priority        INT,
    attachment_id  INT,
    OUT SQLMessage  VARCHAR( 1000 ),
    OUT result      INT)

```


Argument	Max Size; Value Range	Description
Telephones	32672	<p>Recipient's phone numbers. Specify the number exactly as you would dial it from your phone number including any dial out numbers, long distance codes, local area codes and so on. Digits in the number can be separated by optional dashes, parenthesis, spaces and other non-digit symbols. For example, 1 (212) 555-6789</p> <p>If you need to send the same message to multiple recipients, use comma to separate multiple phone numbers.</p>
Message	32672	<p>Message description containing optional text you want the computer to speak to the phone call recipient and also containing optional pre-recorded sound files. Use the Attachment_ID argument to specify which pre-recorded sound files or BLOB values containing sound data you want to use with the message. To specify message file position within the message text use a pair of <wav> and </wav> tags. For example,</p> <p><i>Say <wav>Hello.wav</wav> to my friend</i></p> <p>This message text will instruct DB Mail to perform the following processing steps:</p> <ol style="list-style-type: none"> 1. Convert word "Say" and the trailing space character into a WAV file using computer synthesized human voice. 2. Append pre-recorded hello.wav file to the WAV file created in step 1. 3. Convert words "to my friend" and the leading space character into a WAV file using computer synthesized human voice. 4. Append last WAV file created in step 3 to the file created in step 1 and updated in step 2. <p>Here is another example that uses 2 pre-recorded message segments and text segment as a computer synthesized text-to-speech insertion.</p> <p><i><wav>appointment_reminder</wav> 10:00 AM <wav>call_to_cancel.wav</wav></i></p> <p>This message text will instruct DB Mail to perform the following processing steps:</p> <ol style="list-style-type: none"> 1. Convert text "10:00 AM " including leading and trailing spaces into a WAV file using computer synthesized human voice. 2. Concatenate <i>appointment_reminder.wav</i> file with the WAV created in step 2 and then with <i>call_to_cancel.wav</i> file. <p> Tip: If you need to send a message whose text is longer than the maximum size of varchar data-type in your database you can use a special attachment file with the name <i>message</i> and no extension. DB Mail will use the contents of that file for the message text. The value of the Message parameter will be ignored in that case. For more information see Error! Not a valid bookmark self-reference. topic.</p>
Priority (optional)	0..2	<p>Message processing priority takes a value of 0, 1, or 2. Note that this property is used in 2 different places. If Message Queuing is enabled in DB Mail options, messages having higher priority numbers are processed before messages having lower priority numbers. VoMS server software maintains its own message queue. Message priorities have similar effect on the VoMS server message processing, messages having higher priority numbers are processed before messages having</p>

		lower priority numbers.
Attachment_ID (optional)		ID of the record or group of records in the MAIL_ATTACH table containing attachment data. Message attachment should be created using <i>dbmail.AttachFile</i> and <i>dbmail.AttachData</i> methods. Attached files and data must be in the standard WAV file format.
SQLMessage	1000	Output variable used to return error description in case an error occurs
Result		Output variable to return procedure result

Return values: Returns unique message ID or returns -1 if an error occurs.



Usage Tips:

- One message can have any number of text and sound segments with any positions within the message.
- You may not use nested **<wav>** tags. Nested tags will lead to message processing errors.
- Spaces in the message text have special importance. They are translated into pauses between words and message segments.
- To send a message containing only pre-recorded sound file use simple **<wav>filename.wav</wav>** message text, where *filename.wav* should be replaced with the actual file name.
- When creating pre-recorded sound messages make sure to set sound volume in all files to the same level.

Examples

The following examples demonstrate how to make phone calls and send voice messages using DB Mail procedures.

Example 1 (DB2 SQL; Sending simple dynamically synthesized message to 1 recipient):

The following CALL statement will send "Password expiring" reminder message to telephone number +1 (123) 345-6789. In this example we will dynamically synthesize human voice to speak the entire message.

```
DECLARE Ret INT;
DECLARE ErrMessage VARCHAR(1000);

CALL dbmail.SendVoice( '+1 (123) 345-6789',
    'Attention batch job owners, your database password will
    expire in 3 days. Be sure to update your batch jobs before
    that date. If you need assistance, reply to this message
    with your questions', ErrMessage, Ret);
```

Example 2 (DB2 SQL, Sending dynamically synthesized message from a T-SQL procedure):

This is a more advanced SQL example of a similar use of the *dbmail.SendVoice* procedure:

```

/*****
* This procedure debits the specified account by specified amount
* only if there are sufficient funds to cover the withdrawal, and if there
* are not, it sends an automated phone call to the account holder
* notifying about insufficient funds problem.
* This function returns new account balance.
*****/
CREATE PROCEDURE sp_acct_debit (acct_nbr CHAR(10), debit_amt DECIMAL(11,2))
LANGUAGE SQL
BEGIN
    DECLARE acct_balance DECIMAL(11,2);
    DECLARE ret_code INTEGER;
    DECLARE phone VARCHAR(50);
    DECLARE message VARCHAR(200);
    DECLARE errors VARCHAR(1000);

    SET ret_code = -1;

    SELECT balance
    INTO acct_balance
    FROM accounts
    WHERE acct = acct_nbr;

    IF acct_balance >= debit_amt THEN
        SET acct_balance = acct_balance - debit_amt;

        UPDATE accounts
        SET balance = acct_balance
        WHERE acct = acct_nbr;

        IF @@error = 0 THEN SET @ret_code = 1; END IF; -- success
    ELSE
        -- Insufficient funds.
        -- Send asynchronous voice notification to the account holder
        SELECT phone,
            'Best Bank could not complete your last ' ||
            'debit transaction for account number ending with ' ||
            substr(acct_nbr, 7, 4) + ' because there are ' ||
            'insufficient funds available in your account.'
        INTO phone, message
        FROM accounts
        WHERE acct = acct_nbr;

        CALL dbmail.SendVoice( phone, message, 0, errors, ret_code );

        SET ret_code = 0;
    END;

    RETURN ret_code; -- return code 1 indicates success
                    -- 0 indicates insufficient funds and
                    --      successful notice
                    -- -1 indicates all other problems
END

```

Example 3 (DB2 SQL; Sending 1 message to 1 recipient using pre-recorded sound message stored as external file):

```

DECLARE attach_id INT;
DECLARE Ret INT;
DECLARE ErrMessage VARCHAR(255);

CALL dbmail.AttachFile(NULL, '/files/sound/announcement.wav',

```

```

        errors, attach_id);

CALL dbmail.SendVoice( '+1 (123) 345-6789',
    '<wav>announcement.wav</wav>', 0, attach_id, ErrMessage, Ret )

```

Example 4 (DB2 SQL; Sending 1 message to 1 recipient using pre-recorded sound message stored as internal BLOB in a BLOB data-type column):

```

DECLARE attach_id INTEGER;
DECLARE errors VARCHAR(1000);
DECLARE ret_code INTEGER;
DECLARE data BLOB(1M);

-- read blob value from a table
SELECT wav_data INTO data
FROM sound_files
WHERE doc_type = 'ANNOUNCEMENT' AND doc_key = 1;

-- pass that value to the AttachData procedure
CALL dbmail.AttachData( NULL, 'announce.wav', data,
    errors, attach_id);

-- now, dial out and speak the message
CALL dbmail.SendVoice( '+1 (123) 345-6789', '<wav>announce.wav</wav>', 0,
    attach_id, errors, ret_code );

END

```

Example 5 (Sending multiple-part message containing 1 dynamically synthesized segment and 2 pre-recorded sound segments stored as external files):

```

DECLARE attach_id INTEGER
DECLARE errors VARCHAR(1000);
DECLARE ret INTEGER;

CALL dbmail.AttachFile( NULL, 'c:\misc\dear.wav', errors, attach_id );
CALL dbmail.AttachFile( attach_id, 'c:\campaigns\donate.wav', errors, ret );

CALL dbmail.SendVoice( '+1 (123) 345-6789',
    '<wav>dear.wav</wav> John ' <wav>donate.wav</wav>',
    0, attach_id, errors, ret );

```

CHAPTER 12, Helpful Tips and Recommendations

Performance Tips

Use the following helpful performance tips when sending messages. These tips can greatly improve DB Mail efficiency.

- Do not send large html reports and other large pieces of information within the body of an email message. Instead send them as email attachments.
- If you need to send the same message to multiple recipients, it is more efficient to create just one message and specify multiple recipients as a comma-separated list in the **Recipients** parameter. DB Mail will automatically distribute such message to each listed recipient.
- Email recipient names can include email group names. If you use Microsoft Exchange Server as your email server make sure listed group names are created as email distribution lists. A distribution list may be used for the purpose of sending a single email message to a group of people. This is the most basic and the most common use for lists. Moreover, the Exchange Server can understand a distribution list, which is made up of other distribution lists. For instance, assuming a separate list for each department on campus, a master list can be made up of all those department lists gathered together, eliminating the need to add every faculty/staff member explicitly to the master list.
- If you need to send a message whose text is longer than the maximum size of the **varchar** data-type allowed in your database you can use a special text attachment with the name *MESSAGE* and no extension. DB Mail will use text of this attachment in place of the message text. The value of the **Message** parameter is ignored and superseded by the text of the attachment.



Important note:

When sending email messages with multiple attachments including the special *MESSAGE* attachment, make sure the *MESSAGE* attachment is specified as the last one. Failure to specify *MESSAGE* as the last attachment in the attachment list will lead to incorrect messages.

Usage Tips

- If you want recipients of email message to see their name in the message SENT TO property specify their email address as
name@domain.com<Recipient Name>
- You can design your own fax cover pages using the Fax Cover Page Editor utility available on all Windows 2000 and Windows XP computers. For information on how to create new or modify an existing cover page see [Creating and modifying cover pages](#) topic.
- Fax cover pages are normally stored in the following folder:
- *C:\Documents and Settings\All Users\Start*

Menu\Programs\Accessories\Communications\Fax\My Faxes\Common Coverpages

- Users who want to receive network popup messages and system alerts must be running the Messenger service on their Windows computers.

Database portability tips

DB Mail is designed to provide unified messaging methods in all supported database systems. However, the names of DB Mail methods in Oracle databases differ from names of DB Mail methods in other database management systems. Here is the explanation of why this is done and how to unify DB Mail messaging methods across Oracle and non-Oracle databases.

The DB Mail Oracle implementation conforms to Oracle naming conventions for system built-in packages. Because of this, the names of DB_MAIL package functions in Oracle slightly differ from names of DB Mail functions used in other database management systems. If you are planning to use DB Mail with applications supporting multiple database systems you can create a "proxy" package in Oracle whose only purpose is to provide unified naming across Oracle and non-Oracle databases. Each function of such package will internally call the functionally-matching function in the DB_MAIL package passing through all function parameters.

For example if you are planning to use `dbmail.SendMail` method you can create a new DBMAIL Oracle package as in the following example:

```

/* create DBMAIL package description */
CREATE OR REPLACE PACKAGE system.dbmail IS

    FUNCTION SendMail(
        recipients      VARCHAR2,
        subject         VARCHAR2,
        message         VARCHAR2,
        reply_to        VARCHAR2 DEFAULT NULL,
        content_type    VARCHAR2 DEFAULT 'text/plain',
        priority        INTEGER DEFAULT 1,
        attachment_id   NUMBER DEFAULT NULL)
        RETURN NUMBER;

    FUNCTION AttachFile(
        aid             NUMBER,
        afile_name     VARCHAR2,
        adir            VARCHAR2 )
        RETURN NUMBER;

    FUNCTION AttachData(
        aid             NUMBER,
        afile_name     VARCHAR2,
        adata           BLOB )
        RETURN NUMBER;
END dbmail;
/

show errors

/* create DBMAIL package body */
CREATE OR REPLACE PACKAGE BODY system.dbmail IS
    FUNCTION SendMail(

```

```
    recipients      VARCHAR2,
    subject         VARCHAR2,
    message         VARCHAR2,
    reply_to       VARCHAR2 DEFAULT NULL,
    content_type    VARCHAR2 DEFAULT 'text/plain',
    priority        INTEGER  DEFAULT 1,
    attachment_id   NUMBER    DEFAULT NULL)
RETURN NUMBER
IS
BEGIN
    RETURN db_mail.SendMail( recipients,
                             subject,
                             message,
                             reply_to,
                             content_type,
                             priority,
                             attachment_id);
END SendMail;

FUNCTION AttachFile(
    aid             NUMBER,
    afile_id        NUMBER,
    afile_name      VARCHAR2,
    adir            VARCHAR2 )
RETURN NUMBER
IS
BEGIN
    RETURN db_mail.attach_file( aid,
                                afile_name,
                                adir );
END AttachFile;

FUNCTION AttachData(
    aid             NUMBER,
    afile_name      VARCHAR2,
    adata           BLOB )
RETURN NUMBER
IS
BEGIN
    RETURN db_mail.attach_data( aid,
                                afile_name,
                                adata );
END AttachData;

END dbmail;
/

show errors

/* Create public synonym */
CREATE PUBLIC SYNONYM dbmail FOR system.dbmail
/
```

CHAPTER 13, Troubleshooting and Maintenance

Basic Troubleshooting

You can perform some basic troubleshooting by checking the various directories and files. For example, setting the Diagnostic level in the DB Mail Options screen and then checking the DB_MAIL.LOG DB_MAIL.ERR files in the 'Program Files\DB Mail 2' directory. Usually, you should be able to figure out if a syntax or semantic mistake has been made. Check the [APPENDIX C, Hardware and Software Requirements](#) for further details.

Troubleshooting message processing

To effectively troubleshoot message processing anomalies, set the Diagnostic level in the DB Mail Options screen to **Development**. Reproduce the situation in which you believe messages are not processed correctly and then check the DB_MAIL.LOG file for errors and workings. For more information on supported logging levels and their differences see [Logging level](#) topic.

Troubleshooting database operations

To troubleshoot database operations you can run DB Mail in a special debug mode. To enter this mode start DB Mail from the command line as

```
DB_MAIL /DEBUG
```

A trace window will be displayed for every DB Mail database daemon process and for the DB Mail console. In addition all tracing information will be written to LOG files located in DB Mail home directory.

Also see [Troubleshooting the database connection](#) topic for more information on how to troubleshoot database connections and database operations.

Known Issues

- If fax processing creates a heavy load on the system, the Microsoft Fax Server can periodically hang. As a workaround for this problem you can schedule periodic restart of the server using the following method:
 1. Create a new batch file `RESTART_FAX_SERVER.BAT` containing command to stop/start Fax Server:


```
NET STOP "Fax Server"
NET START "Fax Server"
```
 2. Schedule this file using Windows Control Panel's Add Scheduled Task applet to run every night or day. To add this task from a DOS command prompt use the AT command. For example,


```
AT /EVERY:m,t,w,th,f,s,su 22:00 "C:\RESTART_FAX_SERVER.BAT"
```

which would run it every day each week at 10:00 PM.

For more information on how to schedule a task in Windows please see Microsoft Knowledge Base article [KB300160](#).


APPENDIX A, Starting DB Mail on Computer Startup

To start DB Mail each time Windows starts:

1. Click the Windows **Start** button, and then point to the **Settings**.
2. Click **Taskbar**, and then click the Start Menu **Programs** tab.
3. Click **Add**, and then click **Browse**.
4. Locate DB_MAIL.EXE in the DB Mail installation directory, then double-click it.
5. Click **Next**, and then double-click the **StartUp** folder.
6. Type the name **DB Mail**, which you will see on the **StartUp** menu, and then click **Finish**. Windows will create a shortcut that will be placed to the **SratUp** folder.
7. Repeat steps 1 and 2. Click **Advanced**, and then locate the newly created shortcut.
8. Right-click on the shortcut, and then click **Properties**
9. For the startup window property select **Minimized**, and then click **OK**



Tips:

When DB Mail will start, it will appear as an icon  in the Windows System Tray.

To show the DB Mail Server Console, double-click on the icon or right-click then select **Show** command from the pop-up menu.

If you are running DB Mail on Windows NT or Windows 2000 computer you can also install DB Mail Service. To find out more about it see APPENDIX B, Running DB Mail as a Windows NT service.

APPENDIX B, Running DB Mail as a Windows NT service

DB Mail can be optionally set to run as a Windows NT service. There are several important Windows NT service features that you should know and carefully consider before setting the DB Mail to run as a service:

The DB Mail service can start automatically whenever the computer is started or user is logged to the network and runs continuously in the background.

Use **DB_MAIL.LOG** file to check the service status and activity. This file is located in the same folder as the DB Mail programs – by default it is located in the 'Program Files\DB Mail 2' folder.



Tip:

The DB Mail service is not installed and configured automatically on installation. For installing and configuring the DB Mail service use *Install/Uninstall DB Mail Windows NT Service* shortcuts in the DB Mail program group.

By default the service is installed under LocalSystem account. You should use Control Panel/Services applet to change the DB Mail service account to some other administrative account that has sufficient privileges to access the network and connect to the database. Failure to select the correct account will result in DB Mail being unable to connect to your database servers. **This limitation is due to Windows NT design, for security purposes. Services running under LocalSystem are started before the system is logged to the network and so they do not have network access.**

For more information on Windows NT services, see your Windows NT documentation. You may also want to visit Microsoft technical support on the Web. The following Microsoft knowledge base articles will be useful:

Q124184 - Service Running as System Account Fails Accessing Network.

Q132679 - Local System Account and Null Sessions in Windows NT.

Q158825 - System and User Account Difference

APPENDIX C, Hardware and Software Requirements

DB Mail requires the following minimum hardware and software configurations:

Minimum Hardware Requirements

Front-end:

1. Intel-based or compatible computer
2. At least 64 MB RAM
3. 12 MB disk space
4. Class II fax-modem or better (if fax processing is required on the local computer)
5. VGA monitor

Recommended Configuration

1. Pentium class CPU 400 MHz or better
2. 128 MB RAM or better
3. 18 MB disk space
4. SVGA 256-color or better monitor

Minimum Software Requirements

Back-end:

Any of the supported database servers:

- Oracle 7.3, 8.0, 8i, 9i, 10g
- Microsoft SQL Server 6.5, 7, 2000, 2005
- Sybase SQL Server and Sybase Adaptive Server Enterprise 10.x, 11.x, 12.x
- Sybase Adaptive Server Anywhere 6, 7, 8, 9
- IBM DB2 UDB 5.x, 6.x, 7.x, 8.x



Note: Target Database(s) need not be present in the same system as DB Mail.

Front-end:

1. Windows server or workstation running one of the following operating system:
 - Windows 2003 (server or workstation)
 - Windows XP (server or workstation)
 - Windows 2000 (server or workstation)
 - Windows NT 4.0 (server or workstation with SP4 or better)
2. Required database client software (consult your database system documentation for details)
3. ODBC and ODBC driver (optional; needed if ODBC connection is used)
4. MAPI interface, or Winsock for use with SMTP or Lotus Notes client (installed by default)
5. Fax Printer driver for Fax processing
6. Microsoft Internet Explorer 5.0 or better (required for fax processing)

 **Portability:**

Oracle: Oracle database software is ported to work under different operating systems and is the same on all systems. DB Mail can work with any Oracle database (version 7.3 and later) running on any operating system.

IBM DB2: DB2 database software is ported to work under different operating systems. Most DB2 features required by DB Mail are available in DB2 UDB version 6.1 and later on all systems. Therefore, DB Mail can work with any DB2 database version 6.1 and later running on any Windows, Linux or UNIX system.

Sybase SQL Server and ASE: Sybase SQL Server database software is ported to work under different operating systems. All Sybase SQL Server database features required by DB Mail are available in version 10.0 and later on all systems. Therefore, DB Mail can work with any Sybase database version 10.0 and later running on any operating system.

Sybase ASA: Sybase Adaptive Server Anywhere database software is ported to work under different operating systems. All Sybase ASA database features required by DB Mail are available in version 6.0 and later on all systems. Therefore, DB Mail can work with any Sybase ASA database version 6.0 and later running on any operating system.

Microsoft SQL Server and MSDE: Microsoft SQL Server database software is ported to work under different versions of Windows operating system. All Microsoft SQL Server database features required by DB Mail are available in version 7.0 and later on all systems. Therefore, DB Mail can work with any SQL Server database version 7.0 and later running on any operating system.

APPENDIX D, Technical Support

Your questions, comments, and suggestions are welcome.

For technical support email to support@softtreetech.com or use the on-line support form at <http://www.softtreetech.com/Support.htm>.

Please use the Technical Support Request Form show below when contacting us by email or fax.

When reporting problems, please provide as much information as possible about your problem. Be sure to include the following information:

- 1 Is the problem reproducible? If so, how?
- 2 What version of Windows are you running? For example, Windows XP, Windows NT 4.0, etc.
- 3 What versions of the DB Mail and databases are you running?
- 4 If a dialog box with an error message was displayed, please include the full text of the dialog box, including the text in the title bar.
- 5 If the problem involves an external program, provide as much information as possible on this program?

Also, make sure to include the serial number for your copy of the DB Mail. Use **Help/About** menu to look up the correct numbers. Registered users have priority support.

For registration information, purchasing or other sales information, please contact our sales department sales@softtreetech.com .

For general information, software updates, the latest information on known problems, and answers to frequently asked questions visit the DB Mail home page on the Web: <http://www.softtreetech.com/dbmail/>.

We're happy to help in any way we can, but if you're having problems please check the troubleshooting section first to see if your question is answered there.

Technical Support Request Form

FAX TO: +1 212-208-4625

EMAIL TO: support@softtreetech.com

Please include the following contact information:

Name _____

Company _____

Address _____

City, State/Zip _____

Phone _____ Fax _____

Email _____

Best time to reach you _____

Database system information: Server version _____ Server platform _____

SQL*Net version _____ DB-Lib version _____ CT-Lib version _____

ODBC driver vendor _____ version _____

Which operating system are you using: MS Windows 9x _____ MS Windows NT/2000/XP _____

Email system Information: MAPI _____ SMTP _____ Lotus Notes _____

Fax server software version _____

Computer brand and CPU

type: _____

RAM (MB): _____ Speed (Mhz): _____

CPU quantity and type: _____

Video driver: _____

Relevant devices or peripherals: _____

Description of the problem:

What steps have you taken in order to solve the problem:

APPENDIX E, Licensing

DB MAIL SOFTWARE LICENSE

The SOFTWARE PRODUCT is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The SOFTWARE PRODUCT is licensed, not sold.

CAUTION: Loading this software onto a computer indicates your acceptance of the following terms. Please read them carefully.

GRANT OF LICENSE: SoftTree Technologies, Inc. ("SoftTree Technologies") grants you a license to use the software ("Software"). One licensed copy of the Software may be used for as many database connections as many connections you have licensed.

You may make other copies of the Software for backup and archival purposes only. You may permanently transfer all of your rights under this Software LICENSE only in conjunction with a permanent transfer of your validly licensed copy of the product(s).

LICENSE TYPES: The Software and associated add-in components are licensed on a RUN-TIME basis, which means, that for each computer on which the Software is installed, a valid run-time license must exist.

Single Database Server License

This license type permits using the Software on a single computer (a stand-alone computer or a single workstation in a network or a single network server) per license with one database server. The database server can be running on the same or other computer.

Site License

This license type permits installation and execution of the Software on multiple computers within a single physical location (i.e. an office or data center location at a single physical address). It also permits using the software with any number of database servers.

Enterprise License

This license type permits installation and execution of the Software on multiple computers in multiple locations throughout the licensed company's facilities.

RESTRICTIONS: Unregistered versions (shareware licensed copies) of the Software may be used for a period of not more than 30 days. After 30 days, you must either stop using the Software, or purchase a validly licensed copy.

You must maintain all copyright notices on all copies of the Software. You may not sell copies of the Software to third parties without express written consent of SoftTree Technologies and under SoftTree Technologies' instruction.

EVALUATION copies may be distributed freely without charge so long as the Software remains whole including but not limited to existing copyright notices, installation and setup utilities, help files, licensing agreement, In executing such an act as distributing without the similar copyright or license violation, to the maximum extent permitted by applicable law you may be held liable for loss of revenue to SoftTree Technologies or SoftTree Technologies' representatives due to loss of sales or devaluation of the Software or both.

You must comply with all applicable laws regarding the use of the Software.

COPYRIGHT: The Software is the proprietary product of SoftTree Technologies and is protected by copyright law. You acquire only the right to use the Software and do not acquire any rights of

ownership.

For your convenience, SoftTree Technologies provides certain Software components in the source code format. You may customize this code for your environment, but you agree not to publish, transfer, or redistribute in any other form both the original code and the modified code.

You agree not to remove any product identification, copyright notices, or other notices or proprietary restrictions from the Software.

You agree not to cause or permit the reverse engineering, disassembly, or decompilation of the Software. You shall not disclose the results of any benchmark tests of the Software to any third party without SoftTree Technologies' prior written approval.

DESCRIPTION OF OTHER RIGHTS AND LIMITATIONS: You may not rent, lease or transfer the Software except as outlined under GRANT OF LICENSE - use and copy.

Without prejudice to any other rights, SoftTree Technologies may terminate this Software LICENSE if you fail to comply with the terms and conditions of this Software LICENSE. In such event, you must destroy all copies of the Software and all of its component parts.

WARRANTY DISCLAIMER: SoftTree Technologies is providing this license on an "as is" basis without warranty of any kind; SoftTree Technologies disclaims all express and implied warranties, including the implied warranties of merchantability or fitness for a particular purpose.

LIMITATION OF LIABILITY: SoftTree Technologies shall not be liable for any damages, including direct, indirect, incidental, special or consequential damages, or damages for loss of profits, revenue, data or data use, incurred by you or any third party, whether in an action in contract or tort, even if you or any other person has been advised of the possibility of such damages.

SoftTree Technologies, Inc.
Ilyce Ct 62,
Staten Island NY, 10306
USA

Copyright (c) SoftTree Technologies, Inc. 2000-2005 All Rights Reserved

VOMS SOFTWARE LICENSE

The SOFTWARE PRODUCT is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The SOFTWARE PRODUCT is licensed, not sold.

CAUTION: Loading this software onto a computer indicates your acceptance of the following terms. Please read them carefully.

GRANT OF LICENSE: SoftTree Technologies, Inc. ("SoftTree Technologies") grants you a **limited** license to use the software ("Software"). One licensed copy of the Software may be used together with each DB MAIL software license for the purposes of providing voice messaging interface to the DB Mail server software only. It may not be used with other software or as standalone server software.

You may make other copies of the Software for backup and archival purposes only. You may

permanently transfer all of your rights under this Software LICENSE only in conjunction with a permanent transfer of your validly licensed copy of the product(s).

LICENSE TYPES: The Software and associated add-in components are licensed on a RUN-TIME basis, which means, that for each computer on which the Software is installed, a valid run-time license must exist.

Single DB Mail License

This license type permits using the Software on a single computer (a stand-alone computer or a single workstation in a network or a single network server) per license with one DB Mail server. The DB Mail server can be running on the same or other computer.

Site License

This license type permits installation and execution of the Software on multiple computers within a single physical location (i.e. an office or data center location at a single physical address). It also permits using the software with any number of DB Mail servers.

Enterprise License

This license type permits installation and execution of the Software on multiple computers in multiple locations throughout the licensed company's facilities.

RESTRICTIONS: Unregistered versions (shareware licensed copies) of the Software may be used for a period of not more than 30 days. After 30 days, you must either stop using the Software, or purchase a validly licensed copy.

You must maintain all copyright notices on all copies of the Software. You may not sell copies of the Software to third parties without express written consent of SoftTree Technologies and under SoftTree Technologies' instruction.

EVALUATION copies may be distributed freely without charge so long as the Software remains whole including but not limited to existing copyright notices, installation and setup utilities, help files, licensing agreement, In executing such an act as distributing without the similar copyright or license violation, to the maximum extent permitted by applicable law you may be held liable for loss of revenue to SoftTree Technologies or SoftTree Technologies' representatives due to loss of sales or devaluation of the Software or both.

You must comply with all applicable laws regarding the use of the Software.

COPYRIGHT: The Software is the proprietary product of SoftTree Technologies and is protected by copyright law. You acquire only the right to use the Software and do not acquire any rights of ownership.

For your convenience, SoftTree Technologies provides certain Software components in the source code format. You may customize this code for your environment, but you agree not to publish, transfer, or redistribute in any other form both the original code and the modified code.

You agree not to remove any product identification, copyright notices, or other notices or proprietary restrictions from the Software.

You agree not to cause or permit the reverse engineering, disassembly, or decompilation of the Software. You shall not disclose the results of any benchmark tests of the Software to any third party without SoftTree Technologies' prior written approval.

DESCRIPTION OF OTHER RIGHTS AND LIMITATIONS: You may not rent, lease or transfer the Software except as outlined under GRANT OF LICENSE - use and copy.

Without prejudice to any other rights, SoftTree Technologies may terminate this Software LICENSE if you fail to comply with the terms and conditions of this Software LICENSE. In such event, you must destroy all copies of the Software and all of its component parts.

WARRANTY DISCLAIMER: SoftTree Technologies is providing this license on an "as is" basis without

warranty of any kind; SoftTree Technologies disclaims all express and implied warranties, including the implied warranties of merchantability or fitness for a particular purpose.

LIMITATION OF LIABILITY: SoftTree Technologies shall not be liable for any damages, including direct, indirect, incidental, special or consequential damages, or damages for loss of profits, revenue, data or data use, incurred by you or any third party, whether in an action in contract or tort, even if you or any other person has been advised of the possibility of such damages.

SoftTree Technologies, Inc.
Ilyce Ct 62,
Staten Island NY, 10306
USA

Copyright (c) SoftTree Technologies, Inc. 2005 All Rights Reserved