MSM-Activate/API

User’s Guide

Version 4.4

March 1998

Micronetics Design Corporation
Disclaimer

Micronetics Design Corporation makes no representations or warranties with respect to this manual. Further, Micronetics Design Corporation reserves the right to make changes in the specification of the product described within this manual and without obligation of Micronetics Design Corporation to notify any person of such revision or changes.

The software described in this document is furnished under a license by Micronetics Design Corporation and may only be used or copied in accordance with the terms of such license.

This document is copyrighted. Micronetics Design Corporation grants you the right to copy this document only in conjunction with a validly licensed copy of the software and subject to all provisions of the software license agreement.

Copyright © 1998
Micronetics Design Corporation
1375 Piccard Drive
Rockville, Maryland 20850
Telephone: 301-258-2605
Fax: 301-840-8943
E-mail: info@micronetics.com
WWW: www.micronetics.com
# MSM-Activate/Java

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>87</td>
</tr>
<tr>
<td>MSM API Class and Methods</td>
<td></td>
</tr>
<tr>
<td>msm_login Method</td>
<td>88</td>
</tr>
<tr>
<td>msm_login2 Method</td>
<td>89</td>
</tr>
<tr>
<td>msm_login3 Method</td>
<td>90</td>
</tr>
<tr>
<td>msm_do Method</td>
<td>91</td>
</tr>
<tr>
<td>msm_do2 Method</td>
<td>91</td>
</tr>
<tr>
<td>msm_xecute Method</td>
<td>92</td>
</tr>
<tr>
<td>msm_logout Method</td>
<td>93</td>
</tr>
<tr>
<td>msm_version Method</td>
<td>93</td>
</tr>
<tr>
<td>msm_keepalive Method</td>
<td>93</td>
</tr>
<tr>
<td>Supporting Classes</td>
<td></td>
</tr>
<tr>
<td>CharRef Class</td>
<td>94</td>
</tr>
<tr>
<td>DO2PARMS Class</td>
<td>94</td>
</tr>
<tr>
<td>IntRef Class</td>
<td>94</td>
</tr>
<tr>
<td>StrRef Class</td>
<td>94</td>
</tr>
<tr>
<td>The Bean</td>
<td></td>
</tr>
<tr>
<td>Methods</td>
<td>95</td>
</tr>
<tr>
<td>Properties</td>
<td>95</td>
</tr>
<tr>
<td>Security</td>
<td></td>
</tr>
<tr>
<td>Samples</td>
<td>97</td>
</tr>
<tr>
<td>Result Codes</td>
<td></td>
</tr>
</tbody>
</table>

# MSM-Activate Samples

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>101</td>
</tr>
<tr>
<td>Sample Programs</td>
<td></td>
</tr>
<tr>
<td>Setup Information</td>
<td>102</td>
</tr>
<tr>
<td>TEST (C++)</td>
<td>103</td>
</tr>
<tr>
<td>Line4 (Java)</td>
<td>106</td>
</tr>
<tr>
<td>Sample (Java)</td>
<td>110</td>
</tr>
<tr>
<td>Test (Java)</td>
<td>114</td>
</tr>
<tr>
<td>OLESimp3 and OLESimp4 (Visual Basic)</td>
<td>115</td>
</tr>
<tr>
<td>Detail (Visual Basic)</td>
<td>117</td>
</tr>
<tr>
<td>Sample32 (Visual Basic)</td>
<td>119</td>
</tr>
<tr>
<td>Simple (Visual Basic)</td>
<td>122</td>
</tr>
<tr>
<td>OLEDemo (Delphi)</td>
<td>124</td>
</tr>
</tbody>
</table>

# Appendices

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return Codes</td>
<td>125</td>
</tr>
<tr>
<td>Constants</td>
<td>132</td>
</tr>
</tbody>
</table>

# Index

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM-Activate/API User's Guide</td>
<td>133</td>
</tr>
</tbody>
</table>
Preface

Acknowledgment

Micronetics Standard M (MSM) is an implementation, with extensions, of the ANSI Standard Specification (X11.1-1995) for the Massachusetts General Hospital Utility Multi Programming System (MUMPS). MUMPS was developed by the Laboratory of Computer Science at Massachusetts General Hospital under grant number HS00240 from the National Center for Health Services Research and Development. MUMPS was a trademark of the Massachusetts General Hospital.

Documentation Conventions

The following documentation conventions are used in this manual.

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RETURN</td>
<td>The carriage return key (normally labeled RETURN, ENTER, and so on).</td>
</tr>
<tr>
<td>CTRL+X</td>
<td>The CTRL key pressed at the same time as the X key, where X is any valid key used in combination with the control key.</td>
</tr>
<tr>
<td>&lt;ERROR&gt;</td>
<td>An MSM error message.</td>
</tr>
<tr>
<td>‘val’</td>
<td>In Help messages, ‘val’ is used to indicate that the user can enter the indicated value. The value is entered without the quotes.</td>
</tr>
<tr>
<td>&gt;</td>
<td>The MSM Programmer prompt.</td>
</tr>
<tr>
<td>...</td>
<td>The series of items repeats a user-specified number of times.</td>
</tr>
<tr>
<td>.</td>
<td>Shows a break in a list where consecutive lines have been omitted.</td>
</tr>
<tr>
<td><strong>Bold</strong></td>
<td>Items in a dialogue are shown in <strong>bold</strong> to indicate a user response.</td>
</tr>
</tbody>
</table>
Getting Started

Overview

The Application Programming Interface (API) component of MSM-Activate enables PC and UNIX applications that can call C language functions to access the MSM database and to call M functions and routines. Client applications can communicate with local and remote MSM systems that are accessed via TCP/IP and can access multiple MSM systems simultaneously.

MSM-Server systems from Version 4.3 onward are capable of acting as MSM-Activate servers, provided their host system supports TCP/IP. To communicate with the server, client applications use the Microsoft Windows dynamic link libraries (DLLs) or the UNIX archive libraries that are supplied with MSM-Activate/API.

If your client environment supports ActiveX controls or COM objects (formerly known as OLE objects) then MSM-Activate/COM provides a simpler interface and more functionality than MSM-Activate/API. Similarly, client environments supporting Java can benefit from MSM-Activate/Java. The MSM-Activate/API interface provides low-level access to MSM-Server systems. Future enhancements are more likely to be made to the MSM-Activate/COM interface and its Java variant rather than to the API interface.

MSM-Activate/API for PC and UNIX

MSM-Activate/API is available for Microsoft Windows (Intel PC) and for selected UNIX platforms. The platform type refers to the system on which the client application runs. Different libraries are required for each platform. Server routines are included with MSM-Server Version 4.3 and later versions. The server platform does not have to match the client platform.

MSM-Activate is not able to interface with MSM-Servers prior to Version 4.3.
PC Distribution Package Files

The PC distribution package includes the API, COM and Java components. It creates a directory structure on the developer’s system during installation. The following tables describe the files in each subdirectory and their intended use. The location of the MSM-Activate directory depends on where the package was installed. The default location is C:\Program Files\Micronetics\MSM-Activate.

### MSM-Activate

<table>
<thead>
<tr>
<th>Distribution File</th>
<th>Description of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSMA8API.cnt</td>
<td>Contents file for MSM-Activate/API help file.</td>
</tr>
<tr>
<td>MSMA8API.hlp</td>
<td>MSM-Activate/API help file.</td>
</tr>
<tr>
<td>MSMA8COM.cnt</td>
<td>Contents file for MSM-Activate/COM help file.</td>
</tr>
<tr>
<td>MSMA8COM.hlp</td>
<td>MSM-Activate/COM help file.</td>
</tr>
<tr>
<td>ReadMe.txt</td>
<td>README text file that describes additions or changes made to this distribution package after the documentation is complete.</td>
</tr>
<tr>
<td>Uninst.isu</td>
<td>File that is used to manage uninstall of MSM-Activate.</td>
</tr>
</tbody>
</table>

### MSM-Activate\Redist

<table>
<thead>
<tr>
<th>Distribution File</th>
<th>Description of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFCOLEUI.DLL</td>
<td>File may be required on end-user systems for applications using MSMOLE.DLL. Check version numbers with existing copy.</td>
</tr>
<tr>
<td>MSMAPI.DLL</td>
<td>16-bit dynamic link library that contains the API function calls. Enables client communication with single-byte (ANSI) servers and Unicode servers.</td>
</tr>
<tr>
<td>msmapi32.dll</td>
<td>32-bit dynamic link library that contains the API function calls. Enables client communication with single-byte (ANSI) servers and Unicode servers.</td>
</tr>
<tr>
<td>msmapiu.dll</td>
<td>32-bit Unicode dynamic link library that contains the API function calls. Enables Unicode-capable clients (such as Windows NT and Windows 95) to communicate with Unicode servers, using the UTF-8 transport format.</td>
</tr>
<tr>
<td>MSMOLE.DLL</td>
<td>MSM-Activate/COM 16-bit version.</td>
</tr>
<tr>
<td>msmole32.dll</td>
<td>MSM-Activate/COM 32-bit version.</td>
</tr>
<tr>
<td>OLEPRO32.DLL</td>
<td>May be required on end-user systems for applications using MSMOLE32.DLL. Check version numbers with existing copy.</td>
</tr>
<tr>
<td>REGSVR.EXE</td>
<td>Tool for registration of MSMOLE.DLL.</td>
</tr>
<tr>
<td>REGSVR32.EXE</td>
<td>Tool for registration of MSMOLE32.DLL.</td>
</tr>
</tbody>
</table>
**MSM-Activate\Redist\ClientRT**

<table>
<thead>
<tr>
<th>Distribution File</th>
<th>Description of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>README.TXT</td>
<td>README text file containing information about this directory that became available after the product documentation was complete.</td>
</tr>
<tr>
<td>SETUP.EXE</td>
<td>Setup program that installs the redistributable components of MSM-Activate on an end-user’s client system. Before use, copy this program either to a diskette or to a directory named Disk1. Developers who have installed the MSM-Activate product do not need to run SETUP.EXE.</td>
</tr>
</tbody>
</table>

<various setup files>

**MSM-Activate\Samples**

<table>
<thead>
<tr>
<th>Distribution File</th>
<th>Description of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Readme_samples.txt</td>
<td>Instructions for using the samples.</td>
</tr>
</tbody>
</table>

**MSM-Activate\Samples\Cdemo**

The following distribution files are provided:

<table>
<thead>
<tr>
<th>Distribution File</th>
<th>Description of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>APITEST.RTN</td>
<td>TEST.DEF</td>
</tr>
<tr>
<td>ctest.ini</td>
<td>TEST.EXE</td>
</tr>
<tr>
<td>MAINFRM.CPP</td>
<td>TEST.H</td>
</tr>
<tr>
<td>MAINFRM.H</td>
<td>test.ico</td>
</tr>
<tr>
<td>RESOURCE.H</td>
<td>TEST.MAK</td>
</tr>
<tr>
<td>STDAFX.H</td>
<td>TEST.RC</td>
</tr>
<tr>
<td>STDAFX.CPP</td>
<td>test.rc2</td>
</tr>
<tr>
<td>TEST.CPP</td>
<td>Test32.exe</td>
</tr>
</tbody>
</table>

**MSM-Activate\Samples\Delphi**

The following distribution files are provided:

<table>
<thead>
<tr>
<th>Distribution File</th>
<th>Description of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>local.mco</td>
<td>Example .MCO file to be edited for Delphi example to run.</td>
</tr>
<tr>
<td>oledem.dof</td>
<td>Project options file for MSM-Activate Delphi sample.</td>
</tr>
<tr>
<td>oledem.dpr</td>
<td>Project source file for MSM-Activate Delphi sample.</td>
</tr>
<tr>
<td>oledem.res</td>
<td>Project resource file for MSM-Activate Delphi sample.</td>
</tr>
<tr>
<td>oledemo.dcu</td>
<td>Binary form description for MSM-Activate Delphi sample.</td>
</tr>
<tr>
<td>oledemo.dfm</td>
<td>Binary form description for MSM-Activate Delphi sample.</td>
</tr>
<tr>
<td>oledemo.pas</td>
<td>Delphi source code for MSM-Activate sample.</td>
</tr>
</tbody>
</table>
### MSM-Activate\Samples\Java

<table>
<thead>
<tr>
<th>Distribution File</th>
<th>Description of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>CharRef.class</td>
<td>Redistributable class for MSM-Activate/Java.</td>
</tr>
<tr>
<td>DO2PARMS.class</td>
<td>Redistributable class for MSM-Activate/Java.</td>
</tr>
<tr>
<td>IntRef.class</td>
<td>Redistributable class for MSM-Activate/Java.</td>
</tr>
<tr>
<td>MSMAPI.class</td>
<td>Redistributable class for MSM-Activate/Java.</td>
</tr>
<tr>
<td>msmapi.jar</td>
<td>JavaBean file.</td>
</tr>
<tr>
<td>StrRef.class</td>
<td>Redistributable class for MSM-Activate/Java.</td>
</tr>
<tr>
<td>Test.class</td>
<td>Built version of Test.java</td>
</tr>
<tr>
<td>Test.java</td>
<td>Simple Java test program source code.</td>
</tr>
</tbody>
</table>

### MSM-Activate\Samples\Java\line4

<table>
<thead>
<tr>
<th>Distribution File</th>
<th>Description of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Board.class</td>
<td>Related compiled Java file for Sample.</td>
</tr>
<tr>
<td>Bottom.class</td>
<td>Related compiled Java file for Sample.</td>
</tr>
<tr>
<td>CharRef.class</td>
<td>Related compiled Java file for Sample.</td>
</tr>
<tr>
<td>ColSel.class</td>
<td>Related compiled Java file for Sample.</td>
</tr>
<tr>
<td>computer.gif</td>
<td>Image for use on the sample’s HTML page.</td>
</tr>
<tr>
<td>DO2PARMS.class</td>
<td>Redistributable class for MSM-Activate/Java.</td>
</tr>
<tr>
<td>ENGINE.TXT</td>
<td>M routine to be restored onto the MSM-Activate server.</td>
</tr>
<tr>
<td>IntRef.class</td>
<td>Redistributable class for MSM-Activate/Java.</td>
</tr>
<tr>
<td>line4.class</td>
<td>Compiled version of line4 sample.</td>
</tr>
<tr>
<td>line4.html</td>
<td>HTML page invoking this sample.</td>
</tr>
<tr>
<td>line4.java</td>
<td>Java source code of line4 sample.</td>
</tr>
<tr>
<td>line4F.class</td>
<td>Compile stand alone applet support for line4 sample.</td>
</tr>
<tr>
<td>line4F.java</td>
<td>Source code of stand alone applet support for line4 sample.</td>
</tr>
<tr>
<td>MSMAPI.class</td>
<td>Redistributable class for MSM-Activate/Java.</td>
</tr>
<tr>
<td>Skill.class</td>
<td>Related compiled Java file for Sample.</td>
</tr>
<tr>
<td>StrRef.class</td>
<td>Redistributable class for MSM-Activate/Java.</td>
</tr>
<tr>
<td>Top.class</td>
<td>Related compiled Java file for Sample.</td>
</tr>
<tr>
<td>user.gif</td>
<td>Image for use on the sample's HTML page.</td>
</tr>
</tbody>
</table>
### MSM-Activate\Samples\Java\sample

<table>
<thead>
<tr>
<th>Distribution File</th>
<th>Description of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>BArea.class</td>
<td>Related compiled Java file for Sample.</td>
</tr>
<tr>
<td>CharRef.class</td>
<td>Redistributable class for MSM-Activate/Java.</td>
</tr>
<tr>
<td>DO2PARMS.class</td>
<td>Redistributable class for MSM-Activate/Java.</td>
</tr>
<tr>
<td>IntRef.class</td>
<td>Redistributable class for MSM-Activate/Java.</td>
</tr>
<tr>
<td>LArea.class</td>
<td>Related compiled Java file for Sample.</td>
</tr>
<tr>
<td>MSMAPL.class</td>
<td>Redistributable class for MSM-Activate/Java.</td>
</tr>
<tr>
<td>OArea.class</td>
<td>Related compiled Java file for Sample.</td>
</tr>
<tr>
<td>Sample.class</td>
<td>Compiled version of Sample.</td>
</tr>
<tr>
<td>Sample.html</td>
<td>HTML page that invokes this sample.</td>
</tr>
<tr>
<td>Sample.java</td>
<td>Java source code of Sample.</td>
</tr>
<tr>
<td>Sample.mak</td>
<td>Microsoft Visual J++ make-file for this sample.</td>
</tr>
<tr>
<td>SampleF.class</td>
<td>Compiled stand alone applet support for Java Sample.</td>
</tr>
<tr>
<td>SampleF.java</td>
<td>Source code of stand alone applet support for Java Sample.</td>
</tr>
<tr>
<td>StrRef.class</td>
<td>Redistributable class for MSM-Activate/Java.</td>
</tr>
</tbody>
</table>

### MSM-Activate\Samples\Lib

<table>
<thead>
<tr>
<th>Distribution File</th>
<th>Description of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>msmapi.h</td>
<td>Header that contains return codes and defined values for inclusion in C applications.</td>
</tr>
<tr>
<td>MSMAPI.LIB</td>
<td>Static link library for use with applications that require the 16-bit MSMAPI library to be linked at compile-time.</td>
</tr>
<tr>
<td>msmapi32.lib</td>
<td>Static link library for use with applications that require the 32-bit MSMAPI library to be linked at compile-time.</td>
</tr>
<tr>
<td>msmapiu.lib</td>
<td>Static link library for use with applications that require 32-bit access to Unicode servers. Uses the UTF-8 transport format.</td>
</tr>
</tbody>
</table>

### MSM-Activate\Samples\VBDemos

<table>
<thead>
<tr>
<th>Distribution File</th>
<th>Description of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>DETAIL.BAS</td>
<td>Visual Basic module file for the Details example.</td>
</tr>
<tr>
<td>DETAIL.EXE</td>
<td>Executable of the Details Visual Basic example (VB3).</td>
</tr>
<tr>
<td>DETAIL.FRM</td>
<td>Visual Basic form definition for the Details example.</td>
</tr>
<tr>
<td>DETAIL.FRX</td>
<td>Visual Basic binary form file.</td>
</tr>
<tr>
<td>DETAIL.MAK</td>
<td>Visual Basic project file for the Details example.</td>
</tr>
<tr>
<td>DETALINK.FRM</td>
<td>Visual Basic form definition for the Link window in the Details example.</td>
</tr>
<tr>
<td>DETALINK.FRX</td>
<td>Visual Basic binary form file.</td>
</tr>
<tr>
<td>MSMAPI.BAS</td>
<td>Visual Basic module file that contains all MSM-Activate/API function definitions in the 16-bit DLL.</td>
</tr>
<tr>
<td>MSMAPI32.BAS</td>
<td>Visual Basic module file that contains all MSM-Activate/API function definitions in the 32-bit DLL.</td>
</tr>
<tr>
<td>OLESIMP3.EXE</td>
<td>Executable of the Visual Basic 3 MSM-Activate/COM example.</td>
</tr>
<tr>
<td>olesimp3.frm</td>
<td>Visual Basic 3 form for the MSM-Activate/COM example.</td>
</tr>
</tbody>
</table>
## Distribution File Description of Contents

<table>
<thead>
<tr>
<th>Distribution File</th>
<th>Description of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLESIMP3.FRX</td>
<td>Visual Basic binary form file.</td>
</tr>
<tr>
<td>olesimp3.mak</td>
<td>Visual Basic 3 make-file for the MSM-Activate/COM example.</td>
</tr>
<tr>
<td>OLESIMP4.EXE</td>
<td>Executable of the Visual Basic 4 MSM-Activate/COM example.</td>
</tr>
<tr>
<td>olesimp4.frm</td>
<td>Visual Basic 4 form for the MSM-Activate/COM example.</td>
</tr>
<tr>
<td>olesimp4.frx</td>
<td>Visual Basic binary form file.</td>
</tr>
<tr>
<td>olesimp4.vbp</td>
<td>Visual Basic 4 project file for the MSM-Activate/COM example.</td>
</tr>
<tr>
<td>Sample32.Bas</td>
<td>Visual Basic 4 module file for the SAMPLE32 example.</td>
</tr>
<tr>
<td>sample32.exe</td>
<td>32-bit executable of the SAMPLE32 Visual Basic example (VB4).</td>
</tr>
<tr>
<td>sample32.Frm</td>
<td>Visual Basic 4 form definition for the SAMPLE32 example.</td>
</tr>
<tr>
<td>sample32.frm</td>
<td>Visual Basic binary form file.</td>
</tr>
<tr>
<td>sample32.vbp</td>
<td>Visual Basic 4 project file for the SAMPLE32 example.</td>
</tr>
<tr>
<td>SIMPLE.BAS</td>
<td>Visual Basic module file for the Simple example.</td>
</tr>
<tr>
<td>SIMPLE.EXE</td>
<td>Executable of the Simple Visual Basic example (VB3).</td>
</tr>
<tr>
<td>SIMPLE.FRM</td>
<td>Visual Basic form definition for the Simple example.</td>
</tr>
<tr>
<td>SIMPLE.FRX</td>
<td>Visual Basic binary form file.</td>
</tr>
<tr>
<td>SIMPLE.MAK</td>
<td>Visual Basic project file for the Simple example.</td>
</tr>
<tr>
<td>SIMPLINK.FRM</td>
<td>Visual Basic form definition for the Link window in the Simple example.</td>
</tr>
<tr>
<td>SIMPLINK.FRX</td>
<td>Visual Basic binary form file.</td>
</tr>
<tr>
<td>testole.mco</td>
<td>Example MSM-Activate/COM connection file.</td>
</tr>
<tr>
<td>testtcp.mco</td>
<td>Example MSM-Activate/COM connection file.</td>
</tr>
</tbody>
</table>

### UNIX Distribution Package Files

For each file on the UNIX distribution diskette or tape, the following table describes the file and its intended use.

<table>
<thead>
<tr>
<th>Distribution File</th>
<th>Description of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Readme.unix</td>
<td>README file that describes additions or changes made to the distribution package after the documentation was completed.</td>
</tr>
<tr>
<td>libmsmapi4.a</td>
<td>Platform-specific compile-time library that contains the API function calls.</td>
</tr>
<tr>
<td>msmapi.h</td>
<td>Header that contains return codes and defined values for inclusion in C applications.</td>
</tr>
<tr>
<td>gd.c</td>
<td>C example that demonstrates how to retrieve a global directory listing from the MSM-Server.</td>
</tr>
<tr>
<td>glist.c</td>
<td>C example that demonstrates how to list an M global. Note that this example uses low-level functions which may be removed in a future release. New users are encouraged to use the msm_do* or msm_xecute* function.</td>
</tr>
<tr>
<td>test.c</td>
<td>C example that calls several different MSM-Activate functions. This example uses low-level functions that may be removed in a future release. New users are encouraged to use the msm_do* or msm_xecute* function.</td>
</tr>
<tr>
<td>ver_lib.c</td>
<td>C example that returns the version of the MSM-Activate library on the local system.</td>
</tr>
<tr>
<td>ver_mserv.c</td>
<td>C example that returns the version of the MSM-Activate M server routines of the system that was called.</td>
</tr>
</tbody>
</table>
Managing an MSM-Activate Server

This section explains how to use the SYSGEN utility or the APIMGR utility to configure and control an MSM-Activate server system.

The MSM-Activate server routines are provided with MSM-Server Version 4.3 and later versions. These routines enable MSM-Activate to run programs and commands on the server from a remote client that communicates with the server using TCP/IP.

The MSM-Activate server is configured using either SYSGEN or APIMGR. Each configuration in SYSGEN relates to an independent MSM-Activate configuration. The configuration options also are available if you run APIMGR directly, although they always refer to the configuration that is currently running.

Upgrading from Previous Versions

MSM-Activate Version 4.4 provides Unicode support and is fully backward-compatible with earlier versions of MSM-Activate, previously known as MSM-API.

When you upgrade to MSM Version 4.4, the MSM-Activate 4.4 support routines are automatically loaded. Run APIMGR in the server’s MGR UCI to update MSM-Activate data structures. MSM-Activate 4.4 servers also accept connections from MSM-API Version 3 and Version 4 clients.

MSM-Activate 4.4 clients also can connect to the MSM-API Version 4 server routines present on an MSM 4.3 system. However, some of the new features will not be available.

Using SYSGEN

The MSM-Activate server is configured using either SYSGEN or APIMGR. Each configuration in SYSGEN relates to an independent MSM-Activate configuration. You can have one configuration for testing (with security disabled) and another for the live environment (with full security).

To access the MSM-Activate configuration utility from SYSGEN, perform the following steps:

1. Do ^SYSGEN.
2. Select the configuration that is to be edited.
3. Select Network Configuration.
4. Select MSM-Activate Service Configuration.

The configuration options also are available if you run APIMGR directly, although they always refer to the configuration that is currently running.
Configure **MSM-Activate username/passwords**

The username/password combination works differently for Version 3 and Version 4 connections:

- If you enabled Version 3 connections, then you must specify the password for Version 3 clients. This single-text string is passed by the Version 3 client in the login record. If the string matches, the client is allowed full access to the server. To disable password checking for Version 3, enter NONE at the password prompt.

- If you enabled Version 4 connections, the system prompts you to enter a username. For each user, specify an individual password and the security group to which the user belongs. This option supports multiple users with different security permissions.

To delete a user, enter `-<username>` at the username prompt. To disable password checking for an individual user, enter NONE at the password prompt. If no Version 4 users are set up, this disables username and password checking and allows any Version 4 client to connect to the server, regardless of the username or password it provides. In this situation, all clients that connect will be in security group DEFAULT. When a user is defined, username/password checking is defined.

**Username "DEFAULT"**

If the username is left blank (which might occur if MSM-Activate/COM loads an old .MCO file that does not specify the username or if the client software calls the msm_login entry point in the DLL, which also does not specify the username), then the system assigns a username of DEFAULT. If your software does not pass the username to MSM-Activate, you must configure user DEFAULT on the server to allow it to connect (unless no usernames are set up).

**Configure security**

The following security options are provided.

**Allow Version 3 or Version 4 connections**

The MSM-Activate server can service both Version 3- and Version 4-style connections. A Version 3 connection is established by a legacy client application using MSM-API Version 3. Because Version 3 connections are less secure than Version 4 connections, use this option to disallow Version 3 connections if you do not need to support them. By default, the server allows both Version 3 and Version 4 connections.

**Groups**

Each user is assigned to a group. To simplify security management, security options are set for groups rather than for individual users. Enter the name of the group whose properties you wish to create/edit, or enter ^L to obtain a list of the currently defined groups. For each group, you can specify several properties.
Allow Xecute command

If the Xecute command is allowed, the client can make the server do anything that a programmer can do. Although this provides flexibility for the client, it presents a security concern. A user who knows the username and password could use MSM-Activate to kill global variables, delete routines, or alter data.

Rather than allowing new client programs to use the Xecute command, encourage the use of Do commands and extrinsic functions for greater security. This option enables you to disallow the Xecute command for this security group. The default is to disallow Xecute commands. If your existing client program uses this command, you must turn it on.

If you allow Xecute commands, any Do pattern matches that you previously set up will be deleted because the client can Xecute any Do if Xecute commands are allowed. Therefore, it is pointless to restrict the Do entry points.

Allow low-level commands

Many of the MSM-API Version 3 low-level commands (such as read local variable, kill global variable, release lock) are provided in MSM-Activate/API to ensure backward compatibility. These low-level commands are not efficient because there is an overhead for each API call. It is much faster to call a server-side M routine that does all the M processing and then returns the result rather than running low-level commands one-by-one on the server. In addition, low-level commands pose a security risk because they enable you to perform actions such as killing global variables. To avoid unnecessary risk, do not use these commands in new client programs.

Use this option to disallow low-level commands for this security group. The default setting is to disallow low-level commands. If your existing client program uses any of these commands, you must turn on this option.

Pattern match for Do commands

If you have disallowed Xecute commands, you can enter a series of pattern matches to specify whether a particular entry point is to be allowed or disallowed. This feature gives you complete control over which M entry points an Activate client can call and enables you to restrict a client to call only entry points relevant to the application it is running. If no patterns are set up, all entry points are allowed. The patterns are searched sequentially from the lowest index number.

To add or edit an index, enter the index number. To delete an index, enter -<index no>.

To enter an M pattern match to test for, use a standard M ?-style pattern such as .E1 "^R".E (which matches any entry point in a routine beginning with the letter R. Enter the UCIs to which this pattern applies (it can apply to one or all UCIs).

Because most applications run in just one UCI, this entry generally is restricted to the single UCI in which the server application will run. Next, specify whether you want to allow or disallow the command if the match is successful, and whether the server is to search the other patterns or stop after the first match. For example, you can disallow all entry points to all routines beginning with R except for TEST^RIM. If no patterns are matched (but some patterns are specified), the entry point is disallowed. Example of possible pattern matches follow.
Pattern Match Examples for Do Command Security

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1&quot;ENTRY^ROUT&quot;</td>
<td>Allows just the entry point ‘ENTRY^ROUT’.</td>
</tr>
<tr>
<td>.E1&quot;^ROUT&quot;</td>
<td>Allows any entry point in routine ‘ROUT’.</td>
</tr>
<tr>
<td>1U.E1&quot;^ROUT&quot;</td>
<td>Allows any entry point that starts with an uppercase letter in routine ‘ROUT’.</td>
</tr>
<tr>
<td>1U.E1&quot;^RT&quot;.E</td>
<td>Allows any entry point that starts with an uppercase letter in a routine that starts with ‘RT’.</td>
</tr>
</tbody>
</table>

**Configure a different server**

This option allows you to configure multiple MSM-Activate servers running in the same UCI with independent configurations. Use this option if you want multiple servers listening on different TCP/IP ports with different security options.

After selecting this option, enter the number of the server (1-9) to be configured; 1 is the default. After you select a server, all subsequent configuration options apply to the selected server until you exit the program or select a different server. By default, server number 1 is edited when you run APIMGR.

**Configure server ports**

For Version 3 connections, the client must switch TCP/IP port numbers after the initial login and before running any commands. This step is not required for Version 4 connections.

This option enables you to configure the range of TCP/IP port numbers which are used when the port switch takes place. If you only allow Version 4 connections, then you need not configure this option, which is removed from the list.

**Using APIMGR**

The APIMGR utility can be used to control MSM-Activate and to define the configuration that is currently running. APIMGR is located in the MGR UCI on the server. At all prompts in this program, enter ? to display help relevant to this command. At some prompts, you can enter ^L to display a list of selections, such as a list of defined usernames.

Use the SYSGEN utility to manage multiple independent configurations for the MSM-Activate server.

**Enable remote MSM-Activate connections**

Use this option to start the MSM-Activate server listening process and allow MSM-Activate connections. Specify the TCP/IP port on which incoming connections are to be allowed.

**Disable remote MSM-Activate connections**

This option enables you to shut down the MSM-Activate listening process and prevent any additional API connections. If any MSM-Activate servers are running, this option enables you to terminate them as well.
List current servers
Use this option to display whether the MSM-Activate listening process is running and which MSM-Activate servers, if any, are running. For each server, this option displays the job number of the server, the client IP address, TCP/IP port number, timeout value, UCI, VOL, version number of client, and username.

Terminate API server
Use this option to terminate an MSM-Activate server. Either enter the job number of the server to be terminated, or enter ^L to obtain a list of the current servers.

Configuration options
For information on the configuration options in APIMGR, refer to “Configuring MSM-Activate Server.”

Automating MSM-Activate Startup
To automatically start the MSM-Activate service at system startup, run SYSGEN and select the configuration that you want to edit. From the Autostarts and Automounts option, select MSM-Activate Service Autostart.

If you started previous MSM-API versions with an entry in the Automatic Partition Startup table, remove the entry.
Client Application Programming

Overview

To communicate with an MSM-Activate/API server, client applications use function calls that are provided in the MSM-Activate/API libraries. Each function is called with parameters and generates an integer return code, as described in “MSM-Activate/API Reference” in this manual.

Your application uses the #defined return codes supplied in the MSMAPI.H header file. A successful function call returns MSM_SUCCESS, which is defined as zero.

This section explains the communications mechanisms supported by MSM-Activate/API and describes the functions that are used to initialize and terminate connections with local and remote systems.
Connection Management

Before a client program can make MSM-Active/API requests, it must connect to the server MSM system. After a successful connection occurs, other MSM-Activate/API function calls can be made to view or manipulate the database. When the connection is no longer required, the client program must close the connection.

TCP/IP Initialization

If remote connections are required, initialize the underlying TCP/IP stack for use with MSM-Activate/API. The TCP/IP environment is initialized with a call to `msm_initialize`, which must be the first MSM-Activate/API function called. This function, which can be called only once, verifies that the underlying TCP/IP protocol stack is available and is the correct version.

The `msm_initialize` function returns zero (MSM_SUCCESS) if the initialization completes successfully, or it returns an error code if the initialization fails. In this manual, refer to “MSM-Activate/API Reference” for a description of function codes, and to “Return Codes” for a full error code list.

For the PC product, your TCP/IP stack must comply with the WinSock specification which describes a common interface for Windows network applications. WinSock Version 1.1 or later is required.

Establishing a Connection

A connection to an MSM system is established with the `msm_login3` function. A successful call returns a system descriptor that is used by other MSM-Activate/API function calls to determine the MSM system with which it is to communicate. If the function fails, an error code is returned.

Although the `msm_login` function is still supported, use `msm_login3` instead.

To connect to an MSM system using TCP/IP, the host IP address and the TCP port number of the known login server are passed to the `msm_login3` function. Use the APIMGR utility to configure the TCP port number in MSM. The network administrator generally sets the IP address of the host system.

If the login function succeeds, it returns a system descriptor, and a dedicated MSM-Activate server process (%APISERV) is started. The %APISERV process handles all subsequent requests from the client.

Additional Login Parameters

The `msm_login3` function can take additional parameters to modify the default behavior of the connection. These parameters are described below.

Server UCI

This optional parameter specifies in which UCI (user class identifier) the server process should run. If the default UCI (MGR) is required, this parameter is passed as a NULL pointer.
Server volume group

Use this optional parameter in combination with the UCI parameter to identify the UCI/volume group in which the server process runs. If the default volume group (Volume 0) is required, this parameter is passed as a NULL pointer.

Username/password protection

The username and password can be supplied to the msm_login call. Password checking is enabled on the MSM-Server using the APIMGR utility. The password consists of free text, including numerics and spaces. If the password you enter does not match the password required by the MSM system, the login function fails with an MSM_PASSWD error. If password checking is not enabled on the server MSM system, then all usernames and passwords will be allowed.

Timeout period

An optional timeout parameter can be passed to the login function. This parameter determines how long the dedicated %APISERV process waits between MSM-Activate requests. If a request does not occur within the number of seconds specified in this parameter, the server process terminates. If a timeout is not required, the parameter is passed as zero, causing the dedicated server to wait indefinitely for the next request.

Partition size

The login function can take an additional parameter that specifies the size of the MSM partition for the dedicated server process. The partition size is measured in kilobytes and should be in the range of 16 to 16,384, although a typical value is 48.

If your application requires a large local symbol table, modify the partition size to reflect this requirement. A large symbol table is required if many variables are used. If your application requires only the default MSM partition, pass this parameter as zero.

If your MSM-Activate/API functions consistently return with MSM_PGMOV errors, the partition size probably is too small for your application. Future connections should be made with a larger partition size parameter.

Version

On completion, this parameter contains the version string of the Server-side MSM-Activate routines. A client can communicate with either single-byte or Unicode servers by first examining the version string and then switching to the correct DLL for communication.

For example, this parameter contains ‘4.2.0’ if the client is connecting to a server that has MSM-Activate/API Version 4.2.0 routines installed.

Unicode servers add the ‘/U’ extension to the end of the string. A Japanese version of MSM-Server with MSM-Activate/API Version 4.2.0 installed would return ‘4.2.0/U’. A single-byte client receiving this version string could switch to the Unicode DLL (MSMAPIU.DLL) and then reconnect.
Multiple Connections

A client can establish connections to multiple MSM-Server systems by calling msm_login3 with a unique IP address (or the “local” literal) that identifies each system. Each successful connection returns a unique system descriptor for that server. To communicate with a particular system, subsequent MSM-Activate/API function calls must specify the correct descriptor.

Connection Termination and Environment Rundown

To terminate connections and free resources, you must make function calls.

Connection termination

When a connection to an MSM-Server system is not required, the msm_logout function closes the connection. The function call takes the system descriptor as its only parameter, and returns MSM_SUCCESS if it is successful.

Environment rundown

After all connections are terminated and MSM-Activate/API is not required, make a call to msm_rundown. This function deregisters your application from the Windows socket implementation, frees allocated resources, and terminates the local connection server.

If the function is successful, it returns MSM_SUCCESS. If the function is not successful, an error code is returned. Refer to “Return Codes” in this manual for a full error code list.

C Example

/* apierror is a user-supplied error reporting function */
#include "msmapi.h"
char msm_system[] = "183.132.195.214";
int port = 1666;
char uci[]="";
char volgrp[]="";
int return_code;
unsigned int sd;
char username[]="DEFAULT";
char password[]="PEANUTBUTTER";
int timeout=600;
int partsize=128;
char version[32];

/* call msm_initialize to make the WinSock DLL available */
if (return_code = msm_initialize())
        apierror("msm_initialize", return_code);

/* login into 183.132.195.214 returning the system descriptor to sd */
sd = msm_login3(msm_system, port, uci, volgrp, password, timeout, partsize, username, version, 32);

if (sd > MSM_BASE_ERROR)
        apierror("msm_login", sd)
/* is destination a Unicode Server? */
if (strstr(version, "/U"))
{
    /* talking to Unicode */
}

/* function calls to manipulate the MSM database and system 
should be included here */

/* now close down the connection */
if (return_code=msm_logout(sd))
    apierror("msm_logout", return_code);
/* deallocate resources and release instance of WinSock DLL */
if ((return_code=msm_rundown())!= MSM_SUCCESS)
    apierror("msm_rundown", return_code);

Visual Basic Example

' Login into the server returning the system descriptor to sd
Dim ver$
ver$ = Space(32)
sd = msm_login3("183.132.195.214", 1666, "", "", "PEANUTBUTTER", 0, 0, 
"DEFAULT", ver$, 32)
If sd > 10000 Then
    apierror("msm_login", sd)
End If

' Function calls to manipulate the MSM database and system
' should be included here
' Now close down the connection

return_code=msm_logout(sd)
If return_code <> 0 Then
    apierror("msm_logout", return_code)
End If

' Shutdown the MSM local server
return_code=msm_rundown()
if return_code <> 0 Then
    apierror("msm_rundown", return_code)
End If
MSM-Activate/API C Structures

This section discusses the global data block (GDB) and subscript data block (SDB) structures that are used by the MSM-Activate/API data functions to pass global information to and from the server system. Use the MSM-Activate/API functions described in this section to manipulate the GDB and SDB structures.

These structures and functions generally are used for C applications. Visual Basic applications tend to use the MSM-Activate/COM object.

Using MSM-Activate/API C Structures

This section describes the global data block and subscript data block C structures and explains how to use them.

Global Data Block (GDB)

A GDB passes the global name, UCI name, and Volume Group name to or from the application program interface. A GDB is created by declaring a structure of type global_data_block, and is initialized by a call to msm_gdb_initialize.

Use MSM-Activate/API function calls to manipulate a GDB. The first GDB element contains a count of the GDB size in characters. The second element, which can be accessed as gdb.gname_length, contains the length of the name of the global variable. The remaining space in the GDB is used for the names of the global variable, the UCI, and the volume group. Optionally, you can specify the UCI Name and Volume Group Name fields as NULL pointers.

Two additional MSM-Activate/API calls, msm_gdb_insert and msm_gdb_extract, enable you to insert and retrieve information from a GDB.

Subscript Data Block (SDB)

An SDB is used to pass subscripts to and from the application program interface. An SDB is created by declaring a structure of type subscript_data_block and is initialized by a call to msm_sdb_initialize.

Use MSM-Activate/API function calls to manipulate an SDB. The first SDB element contains a count of the SDB size in characters. The second element, which can be accessed as sdb.number_subscripts, contains the number of subscripts currently stored in the SDB and is the only SDB field that you should access directly. The remaining space in the SDB is used for subscripts and overhead information and is reserved for use by the MSM-Activate/API.

Three other calls, msm_sdb_insert, msm_sdb_extract, and msm_sdb_insert_null, allow you to insert and retrieve subscripts from an SDB and to insert an empty string for use by a subsequent msm_global_order.

Structure addressing

Functions that use the GDB and SDB structures require the address of the structures that are to be passed.
Structure definitions

The MSMAPI.H header file includes the following definitions:

#define GDB struct global_data_block
#define SDB struct subscript_data_block

Use one of the following statements to allocate instances of these structures:

1. struct global_data_block gdb
   struct subscript_data_block sdb
2. GDB gdb;
   SDB sdb;

Summary of Functions

The following table lists the MSM-Activate/API functions that are used to manipulate the GDB and SDB structures.

GDB and SDB Manipulation Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msm_gdb_clear</td>
<td>Clears all fields in a GDB.</td>
</tr>
<tr>
<td>msm_gdb_extract</td>
<td>Returns the global name, UCI name, and Volume Group name from a GDB.</td>
</tr>
<tr>
<td>msm_gdb_initialize</td>
<td>Initializes a GDB.</td>
</tr>
<tr>
<td>msm_gdb_insert</td>
<td>Assigns values to the global name, UCI name, and Volume Group name fields in a GDB.</td>
</tr>
<tr>
<td>msm_sdb_clear</td>
<td>Clears the current subscript count in an SDB.</td>
</tr>
<tr>
<td>msm_sdb_count</td>
<td>Returns the subscript count from an SDB.</td>
</tr>
<tr>
<td>msm_sdb_extract</td>
<td>Returns the string value of a single subscript from an SDB.</td>
</tr>
<tr>
<td>msm_sdb_initialize</td>
<td>Initializes an SDB.</td>
</tr>
<tr>
<td>msm_sdb_insert_null</td>
<td>Puts an M empty string into an SDB for use by msm_global_order.</td>
</tr>
<tr>
<td>msm_sdb_insert</td>
<td>Puts a single string subscript into an SDB.</td>
</tr>
</tbody>
</table>

C Example

/* apierror is a user-supplied error reporting function */
#include "msmapi.h"
GDB gdb;
SDB sdb;
int return_code;
/* initialize the GDB & SDB */

if (return_code = msm_gdb_initialize(MSMK_GDB_SIZE, &gdb))
    apierror("msm_gdb_initialize", return_code);
if (return_code = msm_sdb_initialize(MSMK_SDB_SIZE, &sdb))
    apierror("msm_sdb_initialize", return_code);
/* insert the global TESTGLOB, UCI MGR and VG IBM into the global directory block */
if (return_code = msm_gdb_insert("TESTGLOB", &gdb, "MGR", "IBM"))
    apierror("msm_gdb_insert", return_code);
/* insert the subscript "10" as the first subscript and "NAME" as the second subscript in the subscript data block */
if (return_code = msm_sdb_insert(1, "10", &sdb))
    apierror("msm_sdb_insert", return_code);
if (return_code = msm_sdb_insert(2, "NAME", &sdb))
    apierror("msm_sdb_insert", return_code);
MSM-Activate/API Data Functions

The MSM-Activate/API data functions communicate with an MSM-Server system to modify, retrieve, or traverse data. To communicate with an MSM system, the data functions use a system descriptor that is returned from a successful msm_login3 call. The system descriptor is always the first parameter passed to a data function. The remaining parameters are used to send and receive data from the MSM system or to modify the function behavior.

This section describes the MSM-Activate/API data functions and includes examples for C and Visual Basic applications.

Memory Allocation

Functions that return data from the MSM system must have sufficient memory allocated for the parameters that receive the data.

If the data returned from an msm_local_get call requires 30 bytes of memory storage, allocate memory as shown in the following examples. If a string terminator is required, allocate an additional byte.

C Example

```c
char retval[31];
return_code = msm_local_get(sd, "TEST", retval, 31);
```

Visual Basic Example

```vb
retval$ = space(31)
return_code = msm_local_get(sd, "TEST", retval$, 31)
```

Full MSM Error Messages

If an error occurs within the MSM system, the full MSM error can be loaded into the parameter that expects the returned data if two conditions are met: the function has a parameter that can receive data, and sufficient memory (approximately 30 bytes) is allocated for the error message.

Refer to the MSM User's Guide or the MSM Pocket Guide for detailed explanations of full MSM error messages.

Function Categories

In this document, the MSM-Activate/API data functions are subdivided into the following three categories:

- Local variable manipulation functions.
- Global variable manipulation functions.
- The msm_xecute, msm_do, msm_xecute2, and msm_do2 functions.
Local Variable Functions

These functions manipulate local variables in the dedicated MSM-Server partition. Local variables are available only to the connected client and are removed after a connection terminates. Use the following functions to set, retrieve, and remove local variables.

Local Variable Manipulation Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>msm_local_set</code></td>
<td>Assigns a value to an MSM local variable.</td>
</tr>
<tr>
<td><code>msm_local_get</code></td>
<td>Returns the value of an MSM local variable.</td>
</tr>
<tr>
<td><code>msm_local_kill</code></td>
<td>Deletes a variable from the local partition table.</td>
</tr>
</tbody>
</table>

Use these functions to manipulate temporary variables to be used in the same MSM-Activate/API connection period. They are stored in memory and can be accessed quickly.

If many local variables are used, the %APISERV memory partition can become full. If this occurs, MSM-Activate/API functions may generate MSM_PKGMOV return codes, indicating that the memory is full. To release memory, either remove local variables or increase the partition size. The %APISERV process reserves the MSMAPI local variable for internal use.

C Example

```c
/* apierror is a user-supplied error reporting function */
#include "msmapi.h"
unsigned int sd;
int return_code;
char retval[40];

/* log into an MSM-Server returning system descriptor sd */

/* set a local variable TEST equal to "DATA".
   The M equivalent is SET TEST="DATA" */
if (return_code = msm_local_set(sd, "TEST", "DATA"))
   apierror("msm_local_set", return_code);
/* return the contents of local variable test into retval.
   The closest M equivalent is SET retval=TEST */
if (return_code = msm_local_get(sd, "TEST", retval, 40))
   apierror("msm_local_get", return_code);
/* remove the local variable TEST from the partition.
   This is equivalent to KILL TEST */
if (return_code = msm_local_kill(sd, "TEST"))
   apierror("msm_local_kill", return_code);
```

Visual Basic Example

```vb
' Log into an MSM-Server returning system descriptor sd
' set a local variable TEST equal to "DATA".
' The M equivalent is SET TEST="DATA"
return_code = msm_local_set(sd, "TEST", "DATA")
If return_code <> 0 Then
   apierror("msm_local_set", return_code)
End If
' return the contents of local variable test into retval.
' The closest M equivalent is SET retval=TEST
return_code = msm_local_get(sd, "TEST", retval)
If return_code <> 0 Then
   apierror("msm_local_get", return_code)
```
End If

' remove the local variable TEST from the partition.
' This is equivalent to KILL TEST
return_code = msm_local_kill(sd, "TEST")
if return_code <> 0 Then
    apierror("msm_local_kill", return_code)
End If

Global Variable Functions

Global variable functions modify or retrieve data from global variables on an MSM-Server system. Global variables are the main data storage mechanism for an MSM system and are stored on disk. Changes to an MSM global remain even after the MSM-Activate connection terminates. Depending on the protection assigned to a global, it may be available to all MSM users.

All global variable functions contain the following data:

- The system descriptor identifies the desired server.
- The global data block (GDB) identifies the global, UCI, and volume group.
- The subscript data block (SDB) identifies the global’s subscript level.

By default, changes to global variables occur in the UCI and VG in which the login server was started. You can specify a different UCI and VG in the global data block when an MSM-Activate/API function call is used to manipulate globals. Refer to the MSM User's Guide for additional information about UCIs and VGs.

The following table describes the MSM-Activate/API global data functions.

### Global Data Manipulation Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msm_global_data</td>
<td>Returns the $DATA value of a global node. For more information, refer to the MSM Language Reference Manual.</td>
</tr>
<tr>
<td>msm_global_order</td>
<td>Returns the subscript value of the next or previous subscript of a global node. For more information, refer to the MSM Language Reference Manual.</td>
</tr>
<tr>
<td>msm_global_query</td>
<td>Returns the next defined global node in collating sequence. For more information, refer to the MSM Language Reference Manual.</td>
</tr>
<tr>
<td>msm_global_get</td>
<td>Returns the string value of a global node.</td>
</tr>
<tr>
<td>msm_global_kill</td>
<td>Deletes a global, global subtree, or global node.</td>
</tr>
<tr>
<td>msm_global_lock</td>
<td>Makes a global, subtree, or node unavailable for locking by another user.</td>
</tr>
<tr>
<td>msm_global_set_null</td>
<td>Assigns an M empty string to a global node.</td>
</tr>
<tr>
<td>msm_global_set</td>
<td>Assigns a character string to a global node.</td>
</tr>
<tr>
<td>msm_global_unlock</td>
<td>Makes a global, subtree, or node available for locking by another user.</td>
</tr>
</tbody>
</table>
### C Example

```c
/* apierror is a notional error reporting function */
#include "msmapi.h"
GDB gdb;
SDB sdb;
char retval[100];
int return_code;
int sd;
int ddata;

/* connect to an MSM-Server system and populate the GDB 
   and SDB structures */
/* set the global equal to "TEST DATA" */
if (return_code = msm_global_set(sd, &gdb, &sdb, "TEST DATA"))
    apierror("msm_global_set", return_code);

/* get the data in the M global and store in retval */
if (return_code = msm_global_get(sd, &gdb, &sdb, retval, 100))
    apierror("msm_global_get", return_code);

/* return the MSM \$data of the global into ddata */
if (return_code = msm_global_data(sd, &gdb, &sdb, &ddata))
    apierror("msm_global_data", return_code);

/* return the order of the global, modifying the SDB if a sibling 
   node exists */
if (return_code = msm_global_order(sd, &gdb, &sdb, MSMM_SIBLING, 
                                    retval, 100))
    apierror("msm_global_order", return_code);
```

### MSM_XECUTE Function

The `msm_xecute` and `msm_xecute2` functions execute any line of M code on a connected server by using the M XECUTE command. This flexible function is most useful in calling user routines and functions that return data or update the database. Because it is a potential security concern, consider carefully whether it should be enabled on the server.

To return data into the output parameter, use the M WRITE command in your executed code. This function can perform any other MSM-Activate/API data function by executing the equivalent M code.

The `msm_xecute2` function is identical to the `msm_xecute` function except that it returns the length of the data it has returned in the buffer size parameter. The `msm_xecute2` function can return null characters `$C(0)` in the result. The `msm_xecute` function truncates the result at the first null character.

### MSM_XECUTE restrictions

The following restrictions apply to `msm_xecute` and `msm_xecute2`:

- This function cannot call interactive routines.
- If the data returned is greater than the data storage size of the output parameter, it is truncated at that point and the error code `MSM_INCOMPLETEDATA` is returned.
- The executed code must not operate on the TCP/IP socket device (MSM device 56).
C Example

/* apierror is a user-supplied error reporting function */
#include "msmapi.h"
int return_code;
int sd;
char retval[100];

/* Connect to MSM-Server system */

/* return the result of an MSM extrinsic function to retval */
if (return_code = msm_xecute(sd, "W $$UPDATE^DBASE(10)\", retval, 100))
    apierror("msm_xecute", return_code);
/* return the version number of the host MSM system to retval */
if (return_code = msm_xecute(sd, "$ ZV\", retval, 100))
    apierror("msm_xecute", return_code);

Visual Basic Example

' Assume a connection has already been established with the
' system descriptor stored in sd.
retsize = 100
retval$ = space(retsize)
return_code = msm_xecute2(sd, "W $O(^AAA(123))\", retval$, retsize)
' Note that retsize will now contain the length of data returned
If return_code <> 0 Then
    apierror("msm_xecute", return_code)
End If

MSM_DO Function

The msm_do function invokes an entry point in an M routine on a connected server. This flexible function is most useful in calling user subroutines and extrinsic functions that return data or update the database.

To return data into the output parameter, construct the M code as an extrinsic function and QUIT with a value.

MSM_DO Restrictions

The following restrictions apply to msm_do:

- This function cannot call interactive routines.
- The routine cannot write anything to the current device.
- If the data returned is greater than the data storage size of the output parameter, it is truncated at that point, and the error code MSM_INCOMPLETEDATA is returned.
- The M code must not operate on the TCP/IP socket device (MSM device 56).

C Example

/* apierror is a user-supplied error reporting function */
#include "msmapi.h"
int return_code;
int sd;
char retval[100];

/* Connect to MSM-Server system */

/* return the result of an MSM extrinsic function to retval */
if (return_code = msm_do(sd, "UPDATE^DBASE", "10", retval, 100))
    apierror("msm_do", return_code);
Visual Basic Example

' Assume a connection has already been established with the
' system descriptor stored in sd.
retsize = 100
retval$ = space(retsize)
return_code = msm_do(sd, "VERSION^%APISERV", ",", retval$, retsize)
   If return_code <> 0 Then apierror("msm_do", return_code)
End If

MSM_DO2 Function

The msm_do2 function invokes an entry point in an M routine on a connected server. This flexible function is most useful in calling user subroutines and extrinsic functions that return data or update the database. The enhancement over the msm_do function is that msm_do2 supports the passing of parameters by reference rather than just by value. In addition, parameter length can be 64K in the 16-bit version and 16 MB in the 32-bit version.

Besides returning data in the parameters, you can return data into the output parameter by constructing the M code as an extrinsic function which QUITs with a value.

MSM_DO2 restrictions

The following restrictions apply to msm_do2:

- This function cannot call interactive routines.
- The routine cannot write anything to the current device.
- If the data returned is greater than the data storage size of the output parameter, it is truncated at that point and the error code MSM_INCOMPLETEDATA is returned.
- The M code must not operate on the TCP/IP socket device (MSM device 56).

C Example

#include "msmapi.h"
int return_code;
unsigned int sd;
char mref[] = "tag^routine";
char string1[] = "first";
char string2[] = "second";
char string3[] = "third";
char ret[100];
int len;
int flags[3], insizes[3], outsizes[3];
char *params[3];
DO2PARAMS do2;
do2.do2_size = sizeof(DO2PARAMS);
do2.nParams = 3;
do2.flag = flags;
do2.insize = insizes;
do2.buffsize = outsizes;
do2.params = params;
do2.flag[0] = 1; /* by reference */
do2.flag[1] = 1; /* by reference */
do2.flag[2] = 0; /* by value */
do2.params[0] = string1;
do2.params[1] = string2;
do2.params[2] = string3;
do2.insize[0] = do2.buffsize[0] = strlen(string1);
do2.insize[1] = do2.buffsize[1] = strlen(string2);
ret[0] = '\0';
len = 512;
/* This calls $tag$routine(.string1,.string2,string3)
   with the first two parameters by reference and the third
   by value */
if (return_code = msm_do2(sd, mref, &do2, ret, &len))
   apierror("msm_do2", return_code);
MSM-Activate/API and Visual Basic

This section explains how to incorporate the MSM-Activate/API DLL functions into a Microsoft Visual Basic (VB) application. Some familiarity with Visual Basic is assumed.

The techniques described in this section have been mostly superseded by MSM-Activate/COM, which is supplied as a component of MSM-Activate. It is much easier to use MSM-Activate from Visual Basic through the COM object, rather than programming to the C-style API.

Dynamic Link Libraries

To use a dynamic link library (DLL) in your Visual Basic application, declare the functions in the DLL, either in the general declarations of a form or in a separate module (.BAS file). To access the DLL from all forms and modules, declare the DLL functions in a separate module.

Although you need only declare the functions required by your application, you cannot call a function that is not declared.

Declaring functions

To declare functions, insert declare statements in a module file for each function that you need.

Use the following syntax to create a declare statement:

Declare Function <FUNCTION_NAME> Lib "<LIBRARY_NAME>" Alias "<FUNCTION_ALIAS>" ([PARAMETER_LIST]) As <RETURN_TYPE>

The following table describes the elements of a declare statement.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;FUNCTION_NAME&gt;</td>
<td>The name of the function to be used in your Visual Basic application.</td>
</tr>
<tr>
<td>&lt;LIBRARY_NAME&gt;</td>
<td>The name of the DLL that contains the function.</td>
</tr>
<tr>
<td>&lt;FUNCTION_ALIAS&gt;</td>
<td>The name of the function that is held internally in the DLL.</td>
</tr>
<tr>
<td>[PARAMETER_LIST]</td>
<td>A list of parameters to be passed to the function.</td>
</tr>
<tr>
<td>&lt;RETURN_TYPE&gt;</td>
<td>The type of data returned from the function.</td>
</tr>
</tbody>
</table>

Examples

1. This example demonstrates a declaration of the msm_initialize function, which takes no parameters. You must provide the Alias parameter as an uppercase version of the function name.

   Declare Function msm_initialize Lib "MSMAPI.DLL" Alias "MSM_INITIALIZE" () As Integer

2. In this example, a declaration of msm_local_set() is created. This function requires three parameters:

   Declare Function msm_local_set Lib "MSMAPI.DLL" Alias "MSM_LOCAL_SET"
   (ByVal sd As Integer, ByVal local_var$ As String, ByVal Value$) As Integer
At a minimum, all MSM-Activate/API Visual Basic applications should declare the following functions:

- **msm_initialize**
- **msm_login3**
- **msm_logout**
- **msm_rundown**

For a complete list of the MSM-Activate/API declare statements, refer to the MSMAPI.BAS or MSMAPI32.BAS modules that are supplied in the MSM-Activate/API directory.

### Integers in 16-Bit or 32-Bit Versions

If you use either 32-bit MSM-Activate/API DLL (MSMAPI32.DLL or MSMAPIU.DLL), all the integers that are passed to the functions in the DLL must be 32 bits in size and must be declared as long. For example:

```vbnet
Dim i as Long
```

If you use the 16-bit MSM-Activate/API DLL (MSMAPI.DLL), then all integers that are passed to functions in the DLL must be 16 bits in size and must be declared as integer. For example:

```vbnet
Dim i as Integer
```

To specify a variable as an Integer in Visual Basic, place a `%` (percent) symbol at the end of the variable. To specify the variable as a Long, place an `&` (ampersand) symbol at the end of the name of the variable. For example, `i%` is a 16-bit integer and `i&` is a long.

### Visual Basic Global Variables

Variables in Visual Basic are local to a procedure unless they are declared as global in a module file. Most MSM-Activate/API function calls use a system descriptor to identify the MSM-Server system. Unless all of the MSM-Activate/API function calls are in a single Visual Basic procedure, declare the system descriptor as global.

**Example**

```vbnet
Global sd as Integer
```

### Calling Functions

After the MSM-Activate/API functions are defined, call them with the same procedures used for other Visual Basic functions. Before a function is used to receive data, you must allocate sufficient memory to the `retval` parameter. You then can call the function. Use either the DIM statement or the SPACE function to allocate memory.

When MSM-Activate/API returns a value in a string parameter such as `retval`, it does not convert the value into Visual Basic’s string format because MSM-Activate/API is not notified that the caller was Visual Basic. The string’s length does not change. If the result is shorter than the pre-initialized string, truncate it at the first null character by using code in the following form:

```vbnet
retval$ = Left$(retval$, InStr(retval$, Chr$(0))) - 1
```
As the first example below illustrates, assignment of the return string to an object’s property generally performs the appropriate truncation automatically.

**Examples**

1. In this example, 100 bytes of memory are allocated to variable `retval`.
   
   ```vba
   Dim retval As String * 100
   ```

2. As this example illustrates, you also can allocate memory by using the SPACE function.
   
   ```vba
   retval$ = Space(100)
   ```

3. This example demonstrates how a function call is used to obtain data from the local variable `locvar`, and then to return the data into `retval`. The returned value is used to modify the text property of a Visual Basic textbox control.
   
   ```vba
   retval$ = Space(100)
   rc = msm_local_get(sd, "LOCVAR", retval$, 100)
   If rc = MSM_SUCCESS Then
       text1.text = retval$
   Else
       text1.text = "Error :" + rc
   End If
   ```

**Initializing the TCP/IP Interface**

To connect to a remote MSM-Server, initialize the TCP/IP interface with a successful call to `msm_initialize`. This call is not required for connections to the local MSM system.

**Example**

```vba
rc = msm_initialize()
```
Server Connection from Visual Basic

Use the msm_login3 function to login. If the call is successful, a system descriptor that identifies the dedicated M server process is returned.

C Example

```c
#include "msmapi.h"

/* apierror is a user-supplied error reporting function */

char msm_system[] = "183.132.195.214";
int port = 1666;
char uci[]="";
char volgrp[]="";
int return_code;
unsigned int sd;
char username[]="DEFAULT";
char password[]="PEANUTBUTTER";
int timeout=600;
int partsize=128;
char version[32];

/* call msm_initialize to make the WinSock DLL available */
if (return_code = msm_initialize())
    apierror("msm_initalize", return_code);

/* login into 183.132.195.214 returning the system descriptor to sd */
sd = msm_login3(msm_system, port, uci, volgrp, password, timeout,
    partsize, username, version, 32);

if (sd > MSM_BASE_ERROR)
    apierror("msm_login", sd)
```

Visual Basic Example

This example demonstrates how a connection is made to an MSM-Server at address 192.123.199.14 with a connection management server on port 1666. In this example, the required password is PEANUTBUTTER, and a timeout of 120 is requested. The dedicated server uses the default partition size.

```vbs
tcpaddr$ = "192.123.199.14"
port = 1666
username$ = "DEFAULT"
password$ = "PEANUTBUTTER"
uci$ = "MGR"
v$ = "XXX"
timeout = 120
partsize = 0
version$ = Space(32)
rc = msm_login3(tcpaddr$, port, uci$, v$, password$, timeout, partsize,
    username$, version$, 32)
If rc > MSM_BASE_ERROR Then error.text = "Error : " + rc
End If
```
Global Data Structures

If your application uses global data functions, it must use the global data block (GDB) and subscript data block (SDB) structures. Refer to “MSM-Activate/API Data Functions,” in this manual for a description of the GDB and SDB structures.

To define these C structures in Visual Basic, declare each structure as a string of the required length. A GDB should be 25 characters in length; an SDB should be 600 characters.

To declare structures, use the two global constants, MSMK_GDB_SIZE and MSMK_SDB_SIZE, that are defined in the MSMAPI.BAS example module. If you use the structures in more than one procedure, declare them as global.

Examples

1. This example illustrates how C structures are declared as globals in Visual Basic.

   Global gdb As String * MSMK_GDB_SIZE
   Global sdb As String * MSMK_SDB_SIZE

2. In this example, the MSM-Activate/API structure functions are used to manipulate data in these structures.

   rc = msm_gdb_initialize(MSMK_GDB_SIZE, gdb)
   rc = msm_gdb_insert("TEST", gdb, "MGR", "AAA")

   Note Using the msm_xecute function is often simpler than combining other functions.

3. In this example, a single msm_xecute function is used to set the global ^ABDC("NAME") equal to FRED.

   retlen% = 100
   retval% = space(retlen%)
   return_code = msm_xecute(sd, "S ^ABCD(""NAME"")=""FRED"",
                            retval$, retlen%)
Using MSM-Activate/API with UNIX

This section explains how to compile a client MSM-Activate/API application by using the UNIX MSM-Activate/API library. The section also describes the sample applications that are supplied with the MSM-Activate/API for UNIX distribution package.

Archive Files

The MSM-Activate/API functionality for UNIX is contained in the libmsmapi.a archive file.

Compiling with MSM-Activate/API

To compile the client MSM-Activate/API application, include the MSMAPI.H header file in your source. The source code then can be compiled with the archive file to create the executable (or new library). You can add the archive file to the list of libraries in your “make” file.

Example

This example demonstrates how the compilation of the gd.c example program is supplied:

```
c c gd.c libmsmapi.a –o gd
```

On the Solaris platform, the project must be compiled with the libraries socket and nsl. To compile the file gd.c, use the following line:

```
c c -lnsl -lsocket gd.c libmsmapi.a –o gd
```

On the SCO platform, the project must be compiled with the socket library. For example:

```
c c -lsocket gd.c libmsmapi.a –o gd
```

Example UNIX Applications

The MSM-Activate/API for UNIX distribution package includes two example applications, Global Directory and Global List, that demonstrate several of the available functions. This section explains how to use these examples.

Using the Global Directory Application

The Global Directory or gd.c application uses the msm_xecute function to run the ^%GD M routine and to return a global directory listing of the MSM-Server.

Compiling

Use the following command to compile the gd.c source type. A new gd executable is created.

```
c c gd.c libmsmapi.a –o gd
```
Executing

To run the Global Directory application, the IP address and port number of the MSM-Server are passed as command line arguments to the `gd` executable. All other login parameters are set to NULL pointers or to zeroes in the source code and can be modified if different login options are needed.

Example

The following command returns a global directory listing of the MGR UCI on the MSM-Server:

```
./gd 190.180.170.14 1666
```

Using the Global List application

The Global List or `glist.c` application uses several MSM-Activate/API functions to list an M global that is held on the MSM-Server.

Compiling

Use the following command to compile the `glist.c` source type. A new `glist` executable is created.

```
cc glist.c libmsmapi.a -o glist
```

Executing

To run the Global List application, the IP address and port number of the MSM-Server are passed as command line arguments to the `glist` executable. All other login parameters are set to NULL pointers or to zeroes in the source code and can be modified if different login options are needed.

The program then prompts you to enter the name of the global to list. Enter a global name and press RETURN.

Example

```
./glist 190.180.170.14 1666
Global > TEST
```

If the named global exists, the application lists the global. If the global does not exist, an error message is returned.
MSM-Activate/API Reference

Overview

This section details the supported MSM-Activate/API functions and includes the return values for each MSM function call. Refer to “Return Codes” in this manual for a complete error code list.

Reference Section Syntax

In this section, the MSM-Activate/API functions are described using the following format:

```
function_name()
```

A brief description of the function.

Syntax

The syntax which is used to call the function and parameter descriptions.

Description

A full description of the function.

Return values

A list of possible return codes for the function.

Example

An example of C code which is used to call the function. The example code uses an assumed user-supplied error reporting function, `apierror()`, which is not supplied in the libraries.
Using the TCHAR Parameter

In MSM-Activate/API, the parameter type required depends on the version of MSM-Activate/API DLL that is called. When a single-byte (ANSI) version is used (either MSMAPI.DLL or MSMAPI32.DLL), then interpret ‘TCHAR*’ as ‘char*’. When the Unicode version (MSMAPIU.DLL) is used, then interpret ‘TCHAR*’ as ‘unsigned short*’.

A single-byte version expects that a single byte per character will be sent. Because there are two bytes per character under Unicode, the type ‘unsigned short*’ is used. When the size of the provided buffer must be specified, use the number of characters (not the number of bytes). Under a single-byte version, these are the same; under Unicode, there are twice as many bytes as characters.

For example, the prototype for msm_do is:

```c
int msm_do(int sd, TCHAR* mlineref, TCHAR* mparams, TCHAR* value, int bufsize);
```

Using the C language, the following example illustrates calling into a single-byte version:

```c
char retbuffer[128];
int retcode = msm_do("test\^routine", "param", retbuffer, 128);
```

Using the C language in a Windows environment, the call could be written as follows:

```c
TCHAR retbuffer[128];
int retcode = msm_do(_T("test\^routine"), _T("param"), retbuffer, 128);
```

Under Windows, the ‘TCHAR’ type is already known and no translation to ‘unsigned short’ is required. The Windows environment provides the ability to convert ANSI strings into Unicode strings using the _T("string...") format.

In this reference section, all example code assumes the use of a single-byte DLL. To convert to using a Unicode version, follow the above example. Using MSM-Activate/COM simplifies this process because MSM-Activate/COM automatically communicates with the server using the correct DLL for the MSM-Server type.
msm_do()

Calls a label in an M routine by using the M DO command or $$ extrinsic function call.

Syntax

```c
int msm_do (  
    int sd,  
    TCHAR *mlineref,  
    TCHAR *mparams,  
    TCHAR *result,  
    int bufsize );
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sd</td>
<td>A descriptor that identifies the MSM system with which this function communicates.</td>
</tr>
<tr>
<td>mlineref</td>
<td>An M line reference to call, in tag^rtn format.</td>
</tr>
<tr>
<td>mparams</td>
<td>The actual parameter list for the call.</td>
</tr>
<tr>
<td>result</td>
<td>A buffer that holds the result of the call.</td>
</tr>
<tr>
<td>bufsize</td>
<td>An integer value that indicates the size of the buffer allocated to the result parameter.</td>
</tr>
</tbody>
</table>

Description

This function calls the M line reference that is held in mlineref. If mparams is not a NULL pointer, it is supplied as the argument list to the reference. If the M code QUITs with a value, it is returned in result; otherwise, the returned data is an empty string. You must allocate sufficient memory in the result parameter for the returned data.

Do not use this function to run an interactive MSM routine. Use the function to call routines or functions that return data.

Note: The returned data is truncated if the allocated buffer (*result) is too small to accept it.

Return values

If the function succeeds, it returns MSM_SUCCESS. If the function is not successful, an error code is returned. Because this function can perform any M code, any MSM error can be returned. If an error occurs, the full M error message is returned in value. Refer to “Return Codes” in this manual for a full error code list.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_INCOMPLETEDDATA</td>
<td>The data returned from this call is larger than the space allocated for the return value parameter.</td>
</tr>
</tbody>
</table>
C Example

```c
#include "msmapi.h"
int return_code;
unsigned int sd;
char mref [] = "tag\^routine";
char mparams [] = "33,5H"
char result[MSMK_MAXRETURNLEN];
/* equivalent to S result=$\$tag\^routine(33,5H) */
if (return_code = msm_do(sd, mref, mparams, result, MSMK_MAXRETURNLEN))
    apierror("msm_do", return_code);
```

Visual Basic Examples

```vb
retsize% = 512
result$  = SPACE(retsize)
return_code = msm_do(sd, "tag\^routine", "33,5H", result$, retsize%)
if return_code <> 0
    apierror("msm_do", return_code)
End If
To call a function that takes no parameters, you must pass a NULL pointer for
mparams. In Visual Basic, the constant vbNullString can be used for this purpose.
For example:
```
retsize% = 512
result$  = SPACE(retsize)
return_code = msm_do(sd, "tag2\^routine", vbNullString, result$, retsize%)
if return_code <> 0
    apierror("msm_do", return_code)
End If
```
### msm_do2()

Calls a label in an M routine by using the M DO command or $\$ extrinsic function call.

#### Syntax

```c
int msm_do2 (  
    int sd,  
    TCHAR *mlineref,  
    DO2PARAMS *pParams,  
    TCHAR *result,  
    int *bufsize );
```

#### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sd</td>
<td>A descriptor that identifies the MSM system with which this function communicates.</td>
</tr>
<tr>
<td>mlineref</td>
<td>An M line reference to call, in tag^rtn format.</td>
</tr>
<tr>
<td>pParams</td>
<td>The actual parameter list for the call, structure defined below.</td>
</tr>
<tr>
<td>result</td>
<td>A buffer that holds the result of the call.</td>
</tr>
<tr>
<td>bufsize</td>
<td>The address of an integer value that indicates the size of the buffer allocated to the result parameter. On return, this value will contain the actual size of the data returned in result.</td>
</tr>
</tbody>
</table>

```c
struct do2_params_block  
{
    int do2_size; /* size of this structure */
    int nParams;  /* number of parameters */
    int *flag;  /* bit 0=by value/by reference, */
    bit 1=large object,]
    bit 2=array */
    int *insize; /* length of parameter to send */
    int *buffsize; /* length allowed for reference return */
    TCHAR **params; /* pointer to array of parameter buffers */
};
```

```c
#define DO2PARAMS struct do2_params_block
```

#### DO2PARAMS Structure Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>do2_size</td>
<td>Size of the given structure, calculated using size of (DO2PARAMS).</td>
</tr>
<tr>
<td>nParams</td>
<td>Number of parameters being provided. The other parameters are pointers to arrays with nParams number of entries.</td>
</tr>
<tr>
<td>Flag</td>
<td>Indicator that specifies which entries are by-reference and which are by-value. Pointer to array with nParams number of entries.</td>
</tr>
<tr>
<td>Insize</td>
<td>Number of characters of provided buffer to send in request. Pointer to array with nParams number of entries.</td>
</tr>
<tr>
<td>Buffsize</td>
<td>Total size of provided buffer for by-reference return value. Pointer to array with nParams number of entries.</td>
</tr>
<tr>
<td>Params</td>
<td>Array of pointers to the provided buffers, one for each parameter. Pointer to array with nParams number of entries.</td>
</tr>
</tbody>
</table>

On return, the buffsize is modified to show the number of characters returned for a by-reference parameter.
Description

This function calls the M line reference that is held in mlineref. If pParams is not an empty string, it is supplied as the argument list to the reference. If the M code QUITs with a value, it is returned in value; otherwise, the returned data is an empty string. You must allocate sufficient memory in the value parameter for the returned data. By setting up a suitable structure, any of the parameters can be passed by reference rather than by value.

Do not use this function to run an interactive MSM routine. Use the function to call functions, routines, or M statements which return data or an empty string.

Note  The returned data is truncated if the allocated buffer (*value) is too small to accept it.

Return values

If the function succeeds, it returns MSM_SUCCESS. If the function is not successful, an error code is returned. Because this function can perform any M code, any MSM error can be returned. If an error occurs, the full M error is returned in value. Refer to “Return Codes” in this manual for a full error code list.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_INCOMPLETEDATA</td>
<td>The data returned from this call is larger than the space allocated for the return value parameter.</td>
</tr>
</tbody>
</table>

C Example

```c
#include "msmapi.h"
int return_code;
unsigned int sd;
char mref[] = "tag^routine";
char string1[] = "first";
char string2[] = "second";
char string3[] = "third";
char ret[100];
int len;
int flags[3], insizes[3], outsizes[3];
char *parms[3];
D02PARAMS do2;
do2.do2_size = sizeof(D02PARAMS);
do2.nParams = 3;
do2.flag = flags;
do2.insize = insizes;
do2.buffsize = outsizes;
do2.params = params;
do2.flag[0] = 1; /* by reference */
do2.flag[1] = 1; /* by reference */
do2.flag[2] = 0; /* by value */
do2.params[0] = string1;
do2.params[1] = string2;
do2.params[2] = string3;
do2.insize[0] = do2.buffsize[0] = strlen(string1);
do2.insize[1] = do2.buffsize[1] = strlen(string2);
ret[0] = '\0';
len = 512;
/* This calls $$tag^routine(.string1,.string2,string3)
   with the first two parameters by reference and the third
   by value */
if (return_code = msm_do2(sd, mref, &do2, ret, &len))
  apierror("msm_do2", return_code);
```
**msm_gdb_clear()**

Clears the fields in a global data block.

**Syntax**

```c
int msm_gdb_clear(
    struct global_data_block *gdb);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gdb</td>
<td>Specifies a pointer to a GDB that is to be cleared by this function.</td>
</tr>
</tbody>
</table>

**Description**

This function modifies the elements in the GDB passed in gdb. The length of the global name is assigned a value of zero, and the UCI Name and Volume Group Name fields are cleared. To prepare a GDB for use as a naked reference, call `msm_gdb_clear()`.

**Return values**

If the function succeeds, it returns MSM_SUCCESS. If the function is not successful, an error code is returned. The most common error codes for this function are listed below. In some situations, other error codes are returned. Refer to “Return Codes” in this manual for a full error code list.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_GDB</td>
<td>The argument is not a valid GDB.</td>
</tr>
</tbody>
</table>

**Example**

```c
#include "msmapi.h"
GDB gdb;
int return_code;
if ((return_code = msm_gdb_clear(&gdb)) != MSM_SUCCESS)
    apierror("msm_gdb_clear", return_code);
```
**msm_gdb_extract()**

Returns values for the global name, UCI name, volume set name, and global name length from a global data block.

**Syntax**

```c
int msm_gdb_extract(
    struct global_data_block  *gdb,
    TCHAR  *gname,
    int  *global_length,
    TCHAR  *uci_name,
    TCHAR  *vg_name);
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gdb</td>
<td>Specifies a pointer to a GDB.</td>
</tr>
<tr>
<td>gname</td>
<td>Specifies a pointer to the array that receives the global name.</td>
</tr>
<tr>
<td>global_length</td>
<td>Specifies the length of the name of the global variable.</td>
</tr>
<tr>
<td>uci_name</td>
<td>Specifies an optional pointer to the string that receives the MSM UCI name. If this argument is not needed, specify it as a NULL pointer in the function call.</td>
</tr>
<tr>
<td>vg_name</td>
<td>Specifies an optional pointer to the string that receives the MSM volume group name. If this argument is not needed, specify it as a NULL pointer in the function call.</td>
</tr>
</tbody>
</table>

**Description**

The global name, UCI name, volume group name, and global name length are copied from the GDB to the storage locations pointed to by this function’s arguments.

**Return values**

If the function succeeds, it returns MSM_SUCCESS. If this function is not successful, an error code is returned. The most common error codes for this function are listed below. In some situations, other error codes are returned. Refer to “Return Codes” in this manual for a full error code list.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_GDB</td>
<td>The argument is not a valid GDB.</td>
</tr>
</tbody>
</table>

**Example**

```c
#include "msmapi.h"
int return_code;
GDB gdb;
char gname[MSMK_NAME_LENGTH];
char uci_name[MSMK_UCI_NAME_LENGTH];
char vg_name[MSMK_VOLGRP_NAME_LENGTH];
... /* populate gdb */
if ((return_code = msm_gdb_extract(&gdb, gname, uci_name, vg_name)) != MSM_SUCCESS)
    apierror("msm_gdb_extract", return_code);
```
msm_gdb_initialize()

Initializes a global data block that was allocated by the calling function.

**Syntax**

```c
int msm_gdb_initialize ( 
    int gdb_size, 
    struct global_data_block *gdb );
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gdb_size</td>
<td>Specifies the size, in characters, of the GDB that is allocated by the</td>
</tr>
<tr>
<td></td>
<td>calling function.</td>
</tr>
<tr>
<td>gdb</td>
<td>A pointer to the GDB that is to be initialized by this function.</td>
</tr>
</tbody>
</table>

**Description**

To initialize a GDB prior to use, call `msm_gdb_initialize()`. The size of the GDB is assigned to the first element of the GDB structure, the global name length (second element) is set to zero, and the internal elements are initialized for use by MSM.

**Return values**

If the function succeeds, it returns MSM_SUCCESS. If the function is unsuccessful, an error code is returned. The most common error codes for this function are listed below. In some situations, other error codes are returned. Refer to “Return Codes” in this manual for a full error code list.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_GDB</td>
<td>The <code>gdb</code> parameter is not a valid GDB.</td>
</tr>
</tbody>
</table>

**Example**

```c
#include "msmapi.h"
int return_code;
int gdb_size = MSMK_GDB_SIZE;
struct global_data_block gdb;
if (return_code = msm_gdb_initialize(gdb_size, &gdb))
    apierror("msm_gdb_initialize", return_code);
```
**msm_gdb_insert()**

Assigns values to the Global Name, Length of Global Name, UCI Name, and Volume Group Name fields in a global data block.

**Syntax**

```c
int msm_gdb_insert (  
    TCHAR *global_name,  
    struct global_data_block *gdb,  
    TCHAR *uci_name,  
    TCHAR *vg_name);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>global_name</td>
<td>Specifies a pointer to a valid M global name (one alphabetic character or the percent (%) character, followed by a maximum of seven alphanumeric characters). Do not begin the global name with the up-arrow (^).</td>
</tr>
<tr>
<td>gdb</td>
<td>Specifies a pointer to the GDB that receives the global information.</td>
</tr>
<tr>
<td>uci_name</td>
<td>Specifies a pointer to an MSM UCI name (three uppercase characters).</td>
</tr>
<tr>
<td>vg_name</td>
<td>Specifies a pointer to an MSM volume group name (three uppercase characters).</td>
</tr>
</tbody>
</table>

**Description**

The global name, UCI name and volume group name are copied to the GDB. The length of the global name (the second integer in the GDB) is assigned the length of the global name pointed to by the `global_name` argument.

If the UCI name is not required, pass `uci_name` as a NULL pointer, so that global references are performed in the M server UCI. If the global is translated, refer to the *MSM Utility Manual* and the *MSM User's Guide* for additional information. If the volume group name is not needed, pass `vg_name` as a NULL pointer.

If the `uci_name` is a NULL pointer, the previous contents of `uci_name` and `vg_name` are cleared. If the `vg_name` is a NULL pointer, the previous contents of the `vg_name` field are cleared.

**Return values**

If the function succeeds, it returns MSM_SUCCESS. If the function is not successful, an error code is returned. The most common error codes for this function are listed below. In some situations, other error codes are returned. Refer to “Return Codes” in this manual for a full error code list.
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_GDB</td>
<td>The argument is not a valid GDB.</td>
</tr>
<tr>
<td>MSM_NAME</td>
<td>The length of the global name is greater than eight characters, or the name is not a valid MSM global name.</td>
</tr>
<tr>
<td>MSM_NOSUCHUCI</td>
<td>The UCI name is not three uppercase characters.</td>
</tr>
<tr>
<td>MSM_NOSUCHVOLGP</td>
<td>The volume group name is not three uppercase characters.</td>
</tr>
</tbody>
</table>

**Example**

```c
#include "msmapi.h"
int return_code;
char gname[MSMK_NAME_LENGTH];
struct global_data_block gdb;
char uci_name[3];
char vg_name[3];

// Login and populate structures & arrays
if (return_code = msm_gdb_insert(gname, &gdb, uci_name, vg_name))
    apierror("msm_gdb_insert", return_code);
```
**msm_global_data()**

Returns the $DATA value of a global node.

**Syntax**

```c
int msm_global_data(
    unsigned int sd,
    struct global_data_block *gdb,
    struct subscript_data_block *sdb,
    int *value);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sd</td>
<td>A descriptor that identifies the connected MSM system with which this function communicates.</td>
</tr>
<tr>
<td>gdb</td>
<td>Specifies a pointer to a GDB. To use the M naked reference construct, pass a GDB with a global name length of zero.</td>
</tr>
<tr>
<td>sdb</td>
<td>Specifies a pointer to an SDB. To access the top node of a global, specify zero for the current subscript count.</td>
</tr>
<tr>
<td>value</td>
<td>Specifies a pointer to an integer that receives the $DATA value of the global node specified by gdb and sdb.</td>
</tr>
</tbody>
</table>

**Description**

This function performs the M $DATA operation and returns the following values in value.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The global node is undefined.</td>
</tr>
<tr>
<td>1</td>
<td>The global node has a data value but no descendants.</td>
</tr>
<tr>
<td>10</td>
<td>The global node has no data value but has descendants.</td>
</tr>
<tr>
<td>11</td>
<td>The global node has a data value and descendants.</td>
</tr>
</tbody>
</table>

**Return values**

If the function succeeds, it returns MSM_SUCCESS. If the function is unsuccessful, an error code is returned. The most common error codes for this function are listed below. In some situations, other error codes are returned. Refer to “Return Codes” in this manual for a full error code list.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_GDB</td>
<td>The argument is not a valid GDB.</td>
</tr>
<tr>
<td>MSM_SDB</td>
<td>The argument is not a valid SDB.</td>
</tr>
<tr>
<td>MSM_INVSUBSCR</td>
<td>An invalid subscript. The total length of the global reference is greater than that allowed by MSM.</td>
</tr>
<tr>
<td>MSM_PROT</td>
<td>A global protection violation.</td>
</tr>
<tr>
<td>MSM_NAKEDERR</td>
<td>Incorrect use of a naked reference.</td>
</tr>
<tr>
<td>MSM_NAME</td>
<td>The length of the global name in the GDB and the subscript count in the SDB are both zero.</td>
</tr>
</tbody>
</table>
**msm_global_get()**

Returns the value of a global node as a character string.

**Syntax**

```c
int msm_global_get(
    unsigned int sd,
    struct global_data_block *gdb,
    struct subscript_data_block *sdb,
    TCHAR *value,
    int bufsize);
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sd</td>
<td>A descriptor that identifies the connected MSM system with which this function communicates.</td>
</tr>
<tr>
<td>Gdb</td>
<td>Specifies a pointer to a GDB. To use the M naked reference construct, pass a GDB with a global name length of zero.</td>
</tr>
<tr>
<td>Sdb</td>
<td>Specifies a pointer to an SDB. If the global name length in the passed GDB is zero, the SDB must contain at least one subscript. To access the top node of a global, specify zero for the current subscript count.</td>
</tr>
<tr>
<td>Value</td>
<td>Specifies a pointer to the character that receives the value of the global node.</td>
</tr>
<tr>
<td>Bufsize</td>
<td>A integer value that indicates the size of the buffer allocated to the value parameter.</td>
</tr>
</tbody>
</table>

**Description**

This function retrieves the data stored in the M global variable that is referenced by `gdb` and `sdb`. The data is returned into the `value` buffer, which must have sufficient memory allocated before the function is called.

**Return values**

If the function succeeds, it returns MSM_SUCCESS. If the function is not successful, an error code is returned. The most common error codes for this function are listed below. In some situations, other error codes are returned. Refer to “Return Codes” in this manual for a full error code list.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_GDB</td>
<td>The argument is not a valid GDB.</td>
</tr>
<tr>
<td>MSM_SDB</td>
<td>The argument is not a valid SDB.</td>
</tr>
<tr>
<td>MSM_INVSUBSCR</td>
<td>An invalid subscript. The total length of the global reference is greater than that allowed by MSM.</td>
</tr>
<tr>
<td>MSM_PROT</td>
<td>Global protection violation.</td>
</tr>
<tr>
<td>MSM_NAKEDERR</td>
<td>Incorrect use of a naked reference.</td>
</tr>
<tr>
<td>MSM_NAME</td>
<td>The length of the global name in the GDB and the subscript count in the SDB are both zero.</td>
</tr>
<tr>
<td>MSM_NULLSUBSCR</td>
<td>Empty subscript supplied.</td>
</tr>
<tr>
<td>MSM_UNDEF</td>
<td>An undefined global node.</td>
</tr>
</tbody>
</table>
Example

```c
#include "msmapi.h"
unsigned int sd;
int return_code;
struct global_data_block gdb;
struct subscript_data_block sdb;
char value[MSMK_MAXRETURNLEN];

. Login and populate the gdb & sdb structures
.
if (return_code = msm_global_get(sd, &gdb, &sdb, value,
                                MSMK_MAXRETURNLEN))
    apierror("msm_global_get", return_code);
 msm_global_kill()

Deletes a global node.

Syntax

```c
int msm_global_kill (  
    unsigned int  sd,  
    struct global_data_block  *gdb,  
    struct subscript_data_block  *sdb );
```

Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sd</td>
<td>A descriptor that identifies the connected MSM system with which this function communicates.</td>
</tr>
<tr>
<td>gdb</td>
<td>Specifies a pointer to a GDB. To use the M naked reference construct, pass a GDB with a global name length of zero.</td>
</tr>
<tr>
<td>sdb</td>
<td>Specifies a pointer to an SDB. If the global name length in the passed GDB is zero, the SDB must contain at least one subscript. To access the top node of a global, specify zero for the current subscript count.</td>
</tr>
</tbody>
</table>

Description

This function performs the same operation as the M KILL command. The effect depends on whether you specify subscripts in the SDB. If the current subscript count in the SDB is zero, the entire global is deleted; if it is greater than zero, the specified global node is deleted, as are any descendants.

Return values

If the function succeeds, it returns MSM_SUCCESS. If the function is not successful, an error code is returned. The most common error codes for this function are listed below. In some situations, other error codes are returned. Refer to “Return Codes” in this manual for a full error code list.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_GDB</td>
<td>The argument is not a valid GDB.</td>
</tr>
<tr>
<td>MSM_SDB</td>
<td>The argument is not a valid SDB.</td>
</tr>
<tr>
<td>MSM_INV_SUBSCR</td>
<td>An invalid subscript. The total length of the global reference is greater than that allowed by MSM.</td>
</tr>
<tr>
<td>MSM_PROT</td>
<td>Global protection violation.</td>
</tr>
<tr>
<td>MSM_NAKEDERR</td>
<td>Incorrect use of a naked reference.</td>
</tr>
<tr>
<td>MSM_NAME</td>
<td>The length of the global name in the GDB and the subscript count in the SDB are both zero.</td>
</tr>
</tbody>
</table>
Example

```c
#include "msmapi.h"
unsigned int sd;
int return_code;
struct global_data_block gdb;
struct subscript_data_block sdb;

// Login and populate the gdb & sdb structures
if (return_code = msm_global_kill(sd, &gdb, &sdb))
    apierror("msm_global_kill", return_code);
```
**msm_global_lock()**

Locks a global node.

**Syntax**

```c
int msm_global_lock(
    unsigned int sd,
    int mode_flag,
    struct global_data_block *gdb,
    struct subscript_data_block *sdb,
    int max_time);
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sd</td>
<td>A descriptor that identifies the connected MSM system with which this function communicates.</td>
</tr>
<tr>
<td>mode_flag</td>
<td>Specifies the type of lock to perform.</td>
</tr>
<tr>
<td>gdb</td>
<td>Specifies a pointer to a GDB. Because naked references are not allowed, the global name length field must be greater than zero.</td>
</tr>
<tr>
<td>sdb</td>
<td>Specifies a pointer to an SDB. To access the top node of a global, specify zero for the current subscript count.</td>
</tr>
<tr>
<td>max_time</td>
<td>Specifies (in seconds) how long this function attempts to lock a global node. If a timeout is not required, set this value to -1 and the lock attempt will wait until it completes.</td>
</tr>
</tbody>
</table>

**Description**

Use this function to lock global nodes. The function is used in one of three modes, depending on the value of the `mode_flag` parameter. Refer to the following table.

**mode_flag Values**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSMK_LOCK</td>
<td>0</td>
<td>All previous locks are released before the current lock is performed. This is equivalent to the M non-incremental LOCK.</td>
</tr>
<tr>
<td>MSMK_LOCK_INCR</td>
<td>1</td>
<td>The lock count is incremented by one. This is equivalent to the M LOCK + command.</td>
</tr>
<tr>
<td>MSMK_LOCK_ZALL</td>
<td>2</td>
<td>This is equivalent to the MSM ZALLOCATE command.</td>
</tr>
</tbody>
</table>

`msm_global_lock()` prevents another user from locking the same global node until either the global node is explicitly unlocked or the MSM system is closed down. Locking is only a convention and does not physically prevent access to the locked global.

The `max_time` parameter specifies (in seconds) how long this function attempts to lock the global node. If the function is unable to lock the global within the timeout period, the function returns MSM_LTIMEOUT. If a timeout period is not required, set `max_time` to -1.
Return values

If the function succeeds, it returns MSM_SUCCESS. If the function is not successful, an error code is returned. The most common error codes for this function are listed below. In some situations, other error codes are returned. Refer to “Return Codes” in this manual for a full error code list.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_LTIMEOUT</td>
<td>A timeout error. The global could not be locked in the time specified by <code>max_time</code>.</td>
</tr>
<tr>
<td>MSM_GDB</td>
<td>The argument is not a valid GDB.</td>
</tr>
<tr>
<td>MSM_LOCKMODE</td>
<td>The argument is not a valid lock mode.</td>
</tr>
<tr>
<td>MSM_SDB</td>
<td>The argument is not a valid SDB.</td>
</tr>
<tr>
<td>MSM_NAKEDERR</td>
<td>Incorrect use of a naked reference.</td>
</tr>
<tr>
<td>MSM_NULLSUBSCR</td>
<td>Empty subscript supplied.</td>
</tr>
</tbody>
</table>

Example

```c
#include "msmapi.h"
unsigned int sd;
int return_code;
struct global_data_block gdb;
struct subscript_data_block sdb;
int flag = 0;
int timeout = 10;

// Login and populate the gdb & sdb structures

if (return_code = msm_global_lock(sd, flag, &gdb, &sdb, timeout))
    apierror("msm_global_lock", return_code);
```
**msm_global_order()**

Returns the subscript value of the next or previous sibling of a global node in collating sequence.

**Syntax**

```c
int msm_global_order(
    unsigned int sd,
    struct global_data_block *gdb,
    struct subscript_data_block *sdb,
    int option,
    TCHAR *value,
    int bufsize);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sd</td>
<td>A descriptor that identifies the connected MSM system with which this function communicates.</td>
</tr>
<tr>
<td>gdb</td>
<td>Specifies a pointer to a GDB. To use the M naked reference construct, pass a GDB with a global name length of zero.</td>
</tr>
<tr>
<td>sdb</td>
<td>Specifies a pointer to an SDB. The current subscript count in the SDB must be at least one, and the SDB must contain at least one subscript. To specify an empty subscript as an initial value for SORDER, either call msm_sdb_insert_null or insert a rightmost subscript in the SDB with a length of zero.</td>
</tr>
<tr>
<td>option</td>
<td>Specifies an option mask to modify the function’s behavior.</td>
</tr>
<tr>
<td>value</td>
<td>A pointer to the buffer that receives the next subscript. This is not modified if the function returns MSM_NOSIBLING. This argument is optional; if it is not needed, specify it as a NULL pointer.</td>
</tr>
<tr>
<td>bufsize</td>
<td>A integer value that indicates the size of the buffer allocated to the value parameter.</td>
</tr>
</tbody>
</table>

**Description**

This function performs an M SORDER operation with one of three results: it returns the value of the previous or next sibling in collating sequence into `value`, updates `sdb` with this value, or both.

The `option` mask determines how this function operates. Refer to the following table.

**Option Values**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Bit</th>
<th>Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSMM_SIBLING</td>
<td>0</td>
<td>1</td>
<td>Updates the rightmost subscript in the SDB after performing the SORDER. If the return value is the M empty string, the current subscript count is decremented by one and MSM_NOSIBLING is returned.</td>
</tr>
<tr>
<td>MSMM_PREVIOUS</td>
<td>1</td>
<td>2</td>
<td>Returns the subscript value of the previous sibling; this is equivalent to SORDER with a second argument of -1.</td>
</tr>
</tbody>
</table>
If the value of the $ORDER function is the M empty string, this function returns MSM_NOSIBLING, and sdb is not modified.

**Return values**

If the function succeeds, it returns MSM_SUCCESS. If the function is unsuccessful, an error code is returned. The most common error codes for this function are listed below. In some situations, other error codes are returned. Refer to “Return Codes” in this manual for a full error code list.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_NOSIBLING</td>
<td>This warning status is returned if the result of the $ORDER is the M empty string.</td>
</tr>
<tr>
<td>MSM_GDB</td>
<td>The argument is not a valid GDB.</td>
</tr>
<tr>
<td>MSM_SDB</td>
<td>The argument is not a valid SDB.</td>
</tr>
<tr>
<td>MSM_INVSUBSCR</td>
<td>An invalid subscript. The total length of the global reference is greater than that allowed by MSM.</td>
</tr>
<tr>
<td>MSM_PROT</td>
<td>A global protection violation.</td>
</tr>
<tr>
<td>MSM_NAKEDERR</td>
<td>Incorrect use of a naked reference.</td>
</tr>
<tr>
<td>MSM_NAME</td>
<td>The length of the global name in the GDB and the subscript count in the SDB are both zero.</td>
</tr>
</tbody>
</table>

**Example**

```c
#include "msmapi.h"
unsigned int sd;
int return_value;
int option=0;
struct global_data_block gdb;
struct subscript_data_block sdb;
char value[100];
.
. Login and populate structures
.
/* perform $order without modifying sdb */
if (return_value= msm_global_order(sd, &gdb, &sdb, option, value, 100))
    {  
        if (return_value == MSM_NOSIBLING) {
            /* Action for no further nodes in global */
        }
    else
        apierror("msm_global_order",return_value);
    }
/* perform $order and modify the sdb with the result */
option |= 1;
if (return_value= msm_global_order(sd, &gdb, &sdb, option, value, 100))
    {  
        if (return_value == MSM_NOSIBLING) {
            /* Action for no further nodes in global */
        }
    else
        apierror("msm_global_order",return_value);
    }
/* perform reverse $order */
option |= 2;
if (return_value= msm_global_order(sd, &gdb, &sdb, option, value, 100))
    {  
        if (return_value == MSM_NOSIBLING) {
            /* Action for no further nodes in global */
        }
    else
        apierror("msm_global_order",return_value);
    }
```
**msm_global_query()**

Returns the next defined global node to a specified global node in collating sequence.

**Syntax**

```c
int msm_global_query (  
    unsigned int sd,  
    struct global_data_block *gdb,  
    struct subscript_data_block *sdb,  
    struct subscript_data_block *next_sdb );
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sd</td>
<td>A descriptor that identifies the connected MSM system with which this function communicates.</td>
</tr>
<tr>
<td>gdb</td>
<td>Specifies a pointer to a GDB. Because naked references are not permitted, the global name length must be greater than zero.</td>
</tr>
<tr>
<td>sdb</td>
<td>Specifies a pointer to an SDB. The current subscript count in the SDB must be at least one, and the SDB must contain at least one subscript. To specify an empty subscript as an initial value for QUERY, either call msm_sdb_insert_null or insert a rightmost subscript into the SDB with a length of zero.</td>
</tr>
<tr>
<td>next_sdb</td>
<td>Specifies an optional pointer to an SDB that receives the subscript count and values returned by the $QUERY function. If this argument is not needed, specify it as a NULL pointer. The sdb argument receives the count and values.</td>
</tr>
</tbody>
</table>

**Description**

This function performs the M $QUERY operation and updates an SDB with the subscripts of the next defined global node (in collating sequence) to the node originally specified in the SDB. If no global node exists, a zero is returned in the SDB’s current subscript count, and MSM_NOSIBLING is the value of the function return. If the value of the $QUERY function is an M empty string, this function returns MSM_NOSIBLING.

If the next_sdb argument is not a NULL pointer, it receives the results of the $QUERY. If it is a NULL pointer, sdb is modified with the results.

**Return values**

If the function succeeds, it returns MSM_SUCCESS. If the function is not successful, an error code is returned. The most common error codes for this function are listed in the following table. In some situations, other error codes are returned. Refer to “Return Codes” in this manual for a full error code list.
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_NOSIBLING</td>
<td>This warning status is returned if the result of the $QUERY is the empty string.</td>
</tr>
<tr>
<td>MSM_GDB</td>
<td>The argument is not a valid GDB.</td>
</tr>
<tr>
<td>MSM_SDB</td>
<td>The argument is not a valid SDB.</td>
</tr>
<tr>
<td>MSM_INV_SUBSCR</td>
<td>An invalid subscript. The total length of the global reference is greater than that allowed by MSM.</td>
</tr>
<tr>
<td>MSM_PROT</td>
<td>Global protection violation.</td>
</tr>
<tr>
<td>MSM_NAKEDERR</td>
<td>This function does not permit naked references.</td>
</tr>
<tr>
<td>MSM_NAME</td>
<td>The length of the global name in the GDB and the subscript count in the SDB are both zero.</td>
</tr>
</tbody>
</table>

**Example**

```c
#include "msmapi.h"
unsigned int sd;
int return_code;
struct global_data_block gdb;
struct subscript_data_block sdb, next_sdb;
.
. Login and populate structures.
.
if (return_code = msm_global_query(sd, &gdb, &sdb, &next_sdb))
{
    if (return_code == MSM_NOSIBLING) {
        /* function returned an empty string indicating that the end of the global has been reached */
    } else
        apierror("msm_global_query",return_code);
}
```
**msm_global_set()**

Assigns a character string to a global node.

**Syntax**

```c
int msm_global_set(
    unsigned int sd,
    struct global_data_block *gdb,
    struct subscript_data_block *sdb,
    TCHAR *value);
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sd</td>
<td>A descriptor that identifies the connected MSM system with which this function communicates.</td>
</tr>
<tr>
<td>gdb</td>
<td>Specifies a pointer to a GDB. To use the M naked reference construct, pass a GDB with a global name length of zero.</td>
</tr>
<tr>
<td>sdb</td>
<td>Specifies a pointer to an SDB. If the global name length in the passed GDB is zero, the SDB must contain at least one subscript. To access the top node of a global, specify zero for the current subscript count.</td>
</tr>
<tr>
<td>value</td>
<td>Specifies a pointer to a character string that is to be assigned to the global node. The length cannot exceed the maximum MSM global length of the server. A length of zero assigns the M empty string to the global node.</td>
</tr>
</tbody>
</table>

**Description**

This function performs the same operation as the M SET command. The data in `value` is assigned to the global node specified by `gdb` and `sdb`.

**Return values**

If the function succeeds, it returns MSM_SUCCESS. If the function is not successful, an error code is returned. The most common error codes for this function are listed below. In some situations, other error codes are returned. Refer to “Return Codes” in this manual for a full error code list.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_GDB</td>
<td>The argument is not a valid GDB.</td>
</tr>
<tr>
<td>MSM_SDB</td>
<td>The argument is not a valid SDB.</td>
</tr>
<tr>
<td>MSM_INVSUBSCR</td>
<td>An invalid subscript. The total length of the global reference is greater than that allowed by MSM.</td>
</tr>
<tr>
<td>MSM_PROT</td>
<td>Global protection violation.</td>
</tr>
<tr>
<td>MSM_NAKEDERR</td>
<td>Incorrect use of a naked reference.</td>
</tr>
<tr>
<td>MSM_NAME</td>
<td>The length of the global name in the GDB and the subscript count in the SDB are both zero.</td>
</tr>
<tr>
<td>MSM_NULLSUBSCR</td>
<td>Empty subscript supplied.</td>
</tr>
<tr>
<td>MSM_STRLEN</td>
<td>The string value exceeds MSMK_STRING_LENGTH characters.</td>
</tr>
</tbody>
</table>
Example

```c
#include "msmapi.h"
unsigned int sd;
int return_code;
struct global_data_block gdb;
struct subscript_data_block sdb;
char value[] = "test data";

/* Login and populate the gdb & sdb structures */
if (return_code = msm_global_set(sd, &gdb, &sdb, value))
    apierror("msm_global_set", return_code);
```
**msm_global_set_null()**

Assigns an M empty string to a global node.

**Syntax**

```c
int msm_global_set_null (  
    unsigned int sd,  
    struct global_data_block *gdb,  
    struct subscript_data_block *sdb );
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sd</td>
<td>A descriptor that identifies the connected MSM system with which this function communicates.</td>
</tr>
<tr>
<td>gdb</td>
<td>Specifies a pointer to a GDB. To use the M naked reference construct, pass a GDB with a global name length of zero.</td>
</tr>
<tr>
<td>sdb</td>
<td>Specifies a pointer to an SDB. If the global name length in the passed GDB is zero, the SDB must contain at least one subscript. To access the top node of a global, specify zero for the current subscript count.</td>
</tr>
</tbody>
</table>

**Description**

This function assigns an M empty string to a global node specified by gdb and sdb. The function is equivalent to calling msm_global_set with the value parameter equal to the empty string.

**Return values**

If the function succeeds, it returns MSM_SUCCESS. If the function is unsuccessful, an error code is returned. The most common error codes for this function are listed below. In some situations, other error codes are returned. Refer to “Return Codes” in this manual for a full error code list.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_GDB</td>
<td>The argument is not a valid GDB.</td>
</tr>
<tr>
<td>MSM_SDB</td>
<td>The argument is not a valid SDB.</td>
</tr>
<tr>
<td>MSM_INVSUBSCR</td>
<td>An invalid subscript. The global reference length is greater than that allowed by MSM.</td>
</tr>
<tr>
<td>MSM_PROT</td>
<td>Global protection violation.</td>
</tr>
<tr>
<td>MSM_NAKEDERR</td>
<td>Incorrect use of a naked reference.</td>
</tr>
<tr>
<td>MSM_NAME</td>
<td>The length of the global name in the GDB and the subscript count in the SDB are both zero.</td>
</tr>
<tr>
<td>MSM_NULLSUBSCR</td>
<td>Empty subscript supplied.</td>
</tr>
</tbody>
</table>
Example

#include "msmapi.h"
unsigned int sd;
int return_code;
struct global_data_block gdb;
struct subscript_data_block sdb;
.
. Login and populate the gdb & sdb structures
.
if (return_code = msm_global_set_null(sd, &gdb, &sdb))
    apierror("msm_global_set_null", return_code);
msm_global_unlock()

Unlocks a global node.

Syntax

```c
int msm_global_unlock (
    unsigned int  sd,
    int  mode_flag,
    [struct global_data_block  *gdb],
    [struct subscript_data_block  *sdb] );
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sd</td>
<td>A descriptor that identifies the connected MSM system with which this function communicates.</td>
</tr>
<tr>
<td>mode_flag</td>
<td>Specifies the type of unlock to perform.</td>
</tr>
<tr>
<td>gdb</td>
<td>Specifies a pointer to a GDB. Because naked references are not allowed, the global name length field must be greater than zero.</td>
</tr>
<tr>
<td>sdb</td>
<td>Specifies a pointer to an SDB. To access the top node of a global, specify zero for the current subscript count.</td>
</tr>
</tbody>
</table>

Description

Use this function to release global locks. The function is used in one of three modes, depending on the value of the `mode_flag` parameter. Refer to the following table.

**mode_flag Values**

<table>
<thead>
<tr>
<th>mode_flag</th>
<th>Value</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSMK_UNLOCK</td>
<td>0</td>
<td>All previous locks are released. This is equivalent to the M LOCK command without an argument. The global_data_block and subscript_data_block arguments are ignored.</td>
</tr>
<tr>
<td>MSMK_UNLOCK_DECR</td>
<td>1</td>
<td>The lock count is decremented by one. This is equivalent to the M LOCK command.</td>
</tr>
<tr>
<td>MSMK_UNLOCK_ZDEALL</td>
<td>2</td>
<td>This is equivalent to the MSM ZDEALLOCATE command. If the global_data_block and subscript_data_block arguments are NULL pointers, an argumentless ZDEALLOCATE is performed.</td>
</tr>
</tbody>
</table>

Return values

If the function succeeds, it returns MSM_SUCCESS. If the function is not successful, an error code is returned. The most common error codes for this function are listed in the following table. In some situations, other error codes are returned. Refer to “Return Codes” in this manual for a full error code list.
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_GDB</td>
<td>The argument is not a valid GDB.</td>
</tr>
<tr>
<td>MSM_LOCKMODE</td>
<td>The argument is not a valid lock mode.</td>
</tr>
<tr>
<td>MSM_SDB</td>
<td>The argument is not a valid SDB.</td>
</tr>
<tr>
<td>MSM_NAKEDERR</td>
<td>Incorrect use of a naked reference.</td>
</tr>
<tr>
<td>MSM_NULLSUBSCR</td>
<td>Empty subscript supplied.</td>
</tr>
</tbody>
</table>

**Example**

```c
#include "msmapi.h"
unsigned int sd;
int return_code;
struct global_data_block gdb;
struct subscript_data_block sdb;
int flag = 0;

// Login and populate the gdb & sdb structures
if (return_code = msm_global_unlock(sd, flag, &gdb, &sdb))
    apierror("msm_global_unlock", return_code);
```
**msm_initialize()**

Initializes the TCP/IP subsystem for use with MSM-Activate/API.

**Syntax**

```c
int msm_initialize (void);
```

**Description**

If remote TCP/IP connections are used, this function must be the first MSM-Activate/API function to be called by an application. For PC applications, the function verifies that the appropriate version of WINSOCK.DLL is available. MSM-Activate/API requires WinSock Version 1.1 or later.

**Return values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_SUCCESS</td>
<td>Successful completion.</td>
</tr>
<tr>
<td>MSM_TCPLIB</td>
<td>Unable to find WINSOCK.DLL, which should be installed in the Windows directory or one of its subdirectories.</td>
</tr>
<tr>
<td>MSM_TCPVER</td>
<td>WINSOCK.DLL is not Version 1.1 or later.</td>
</tr>
</tbody>
</table>

**Example**

```c
#include "msmapi.h"
int return_code;
if ((return_code = msm_initialize()) != MSM_SUCCESS)
    apierror("msm_initialize", return_code);
```
**msm_local_get()**

Returns the value of an M local variable.

**Syntax**

```c
int msm_local_get (     
    unsigned int sd,   
    TCHAR *local_variable,   
    TCHAR *value,   
    int bufsize);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sd</td>
<td>A descriptor that identifies the connected MSM system with which this function communicates.</td>
</tr>
<tr>
<td>local_variable</td>
<td>The name of the local_variable that contains the data.</td>
</tr>
<tr>
<td>value</td>
<td>A pointer to the buffer that contains the returned contents of the local variable.</td>
</tr>
<tr>
<td>bufsize</td>
<td>An integer value that indicates the size of the buffer that is allocated to the value parameter.</td>
</tr>
</tbody>
</table>

**Description**

This function retrieves the data stored in the M local variable that is referenced in local_variable. The data is returned into the value buffer, which must have sufficient memory allocated before the function is called.

**Return values**

If the function succeeds, it returns MSM_SUCCESS. If the function is not successful, an error code is returned. The most common error codes are listed below. In some situations, other error codes are returned. Refer to “Return Codes” in this manual for a full error code list.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_NAME</td>
<td>An invalid M local variable name.</td>
</tr>
<tr>
<td>MSM_UNDEF</td>
<td>An undefined local variable.</td>
</tr>
<tr>
<td>MSM_INDER</td>
<td>This error is returned if the local_variable is passed as a NULL pointer.</td>
</tr>
</tbody>
</table>

**Example**

```c
#include "msmapi.h"
unsigned int sd;
int return_code;
char local_variable[] = "TEST";
char value[100];
if (return_code = msm_local_get(sd, local_variable, value, 100))
    apierror("msm_local_get", return_code);
```
msm_local_kill()

Deletes an M local variable from the local symbol table.

Syntax

```c
int msm_local_kill (   
    unsigned int sd,   
    TCHAR *local_variable );
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sd</td>
<td>A descriptor that identifies the MSM system with which this function communicates.</td>
</tr>
<tr>
<td>local_variable</td>
<td>The name of the local variable to be deleted.</td>
</tr>
</tbody>
</table>

Description

This function deletes the M local variable that is specified in `local_variable` from the local symbol table.

Return values

If the function succeeds, it returns MSM_SUCCESS. If the function is unsuccessful, an error code is returned. The most common error codes for this function are listed below. In some situations, other error codes are returned. Refer to “Return Codes” in this manual for a full error code list.

If the local variable does not exist, the function returns MSM_SUCCESS.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_NAME</td>
<td>Invalid variable name.</td>
</tr>
</tbody>
</table>

Example

```c
#include "msmapi.h"
unsigned int sd;
int return_code;
char local_variable[] = "TEST"
if (return_code = msm_local_kill(sd, local_variable))
    apierror("msm_local_kill", return_code);
```
**msm_local_set()**

Assigns a value to an M local variable.

**Syntax**

```c
int msm_local_set(
    unsigned int  sd,
    TCHAR  *local_variable,
    TCHAR  *value);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sd</td>
<td>A descriptor that identifies the MSM system with which this function communicates.</td>
</tr>
<tr>
<td>local_variable</td>
<td>The name of the local variable to which the data is assigned.</td>
</tr>
<tr>
<td>value</td>
<td>The data to be assigned.</td>
</tr>
</tbody>
</table>

**Description**

This function performs the same operation as the M SET command on the local variable specified in `local_variable`. The length of the data stored cannot exceed the maximum string length defined as MSMK_STRING_LENGTH. When `value` is passed as a NULL pointer, an M empty string will be assigned to the local variable.

**Return values**

If the function succeeds, it returns MSM_SUCCESS. If the function is unsuccessful, an error code is returned. The most common error codes for this function are listed below. In some situations, other error codes are returned. Refer to “Return Codes” in this manual for a full error code list.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_PGMOV</td>
<td>The local symbol table is full. To resolve this problem, either remove several entries from the partition with <code>msm_local_kill()</code> or increase the partition size with <code>msm_partsiz()</code>.</td>
</tr>
<tr>
<td>MSM_NAME</td>
<td>An invalid M variable name.</td>
</tr>
<tr>
<td>MSM_STRLEN</td>
<td>The value specified is longer than MSMK_STRING_LENGTH.</td>
</tr>
</tbody>
</table>

**Example**

```c
#include "msmapi.h"
unsigned int sd;
int return_code;
char local_variable[] = "TEST";
char value[] = "Test data";
if (return_code = msm_local_set(sd, local_variable, value))
    apierror("msm_local_set", return_code);
```
**msm_login(), msm_login2(), msm_login3()**

Connects to an MSM-Server system.

*msm_login* (which is retained for backward compatibility) does not send a username.

*msm_login2* sends a username.

*msm_login3* enables the client application to know the version number of the MSM-Activate/API server side routines to which it is connecting.

**Syntax**

```c
int msm_login ( 
    TCHAR *server, 
    int port, 
    TCHAR *uci, 
    TCHAR *volgrp, 
    TCHAR *password, 
    int timeout, 
    int partsize );

int msm_login2 ( 
    TCHAR *server, 
    int port, 
    TCHAR *uci, 
    TCHAR *volgrp, 
    TCHAR *password, 
    int timeout, 
    int partsize, 
    TCHAR *username );

int msm_login3 ( 
    TCHAR *server, 
    int port, 
    TCHAR *uci, 
    TCHAR *volgrp, 
    TCHAR *password, 
    int timeout, 
    int partsize, 
    TCHAR *username, 
    TCHAR *version, 
    int versionsize );
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>A pointer to the MSM system address.</td>
</tr>
<tr>
<td>port</td>
<td>The port number of the known login server.</td>
</tr>
<tr>
<td>uci</td>
<td>A pointer to the desired UCI for the server.</td>
</tr>
<tr>
<td>volgrp</td>
<td>A pointer to the desired volume group for the server.</td>
</tr>
<tr>
<td>password</td>
<td>A pointer to the login password.</td>
</tr>
<tr>
<td>timeout</td>
<td>The timeout period for the M server routine.</td>
</tr>
<tr>
<td>partsize</td>
<td>The size (in kilobytes) of the M server process partition.</td>
</tr>
<tr>
<td>username</td>
<td>A pointer to the username to login with.</td>
</tr>
<tr>
<td>version</td>
<td>A pointer to a buffer where the version number of the server-side MSM-Activate will be returned.</td>
</tr>
<tr>
<td>versionsize</td>
<td>The size of the version buffer.</td>
</tr>
</tbody>
</table>

**Description**

Use one of these functions to connect to a specified MSM-Server system.

The `server` parameter identifies the IP address of a remote MSM-Server.

The `port` parameter identifies the TCP port number of the remote known login server.

The `uci` and `volgrp` parameters identify the UCI and volume group in which the server runs. Both `uci` and `volgrp` are three uppercase characters. If the default UCI (MGR) and volume group (volume group 0) are required, these values are passed as NULL pointers. If the UCI or volume group is invalid, the function returns `MSM_NOSUCHUCI` or `MSM_NOSUCHVOLGRP`, respectively. If a volume group is specified and the UCI is a NULL pointer, the function returns `MSM_NOSUCHUCI`.

The `username` and `password` parameters contain an optional login username and password. The M Login routine accepts a connection only if the password is correct for this user. If a password is not required, this parameter is passed as a NULL pointer.

The `timeout` parameter specifies an optional timeout period (in seconds) for the M server process. If this value is other than zero, the M server process shuts down unless it receives another request within `timeout` seconds of the last request. This feature prevents M server processes from continuing to use MSM resources if a client process crashes or fails to call the `msm_rundown()` function.

If the M server process terminates and additional MSM-Activate requests are needed, perform a reconnect by again calling `msm_login2()`. If a timeout period is not needed, set this parameter to zero, thereby programming the M server to wait indefinitely for the next request.

The `partsize` parameter specifies the partition size (in kilobytes) for the M server process. If a large local symbol table is used, adjust this parameter accordingly. If the default partition size for the system is used, set this value to zero.
Return values

If no error occurs, `msm_login()`, `msm_login2()` and `msm_login3()` return a descriptor to a dedicated M server process. This descriptor is used by other MSM-Activate/API function calls to identify the server with which it communicates (parameter `sd` in those function calls).

If an error occurs, one of the following codes is returned. Because the returned descriptor is type `unsigned int`, all return values are positive integers. To differentiate between the valid descriptors and the error codes, the integer value of the error codes begins at 10000 (`MSM_BASE_ERROR` in `MSMAPI.H`).

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_NETINIT</td>
<td>The callable interface was not successfully initialized with <code>msm_initialize()</code>.</td>
</tr>
<tr>
<td>MSM_NETDOWN</td>
<td>The Windows socket implementation detects that the network subsystem failed.</td>
</tr>
<tr>
<td>MSM_SDNUM</td>
<td>No more system descriptors are available.</td>
</tr>
<tr>
<td>MSM_SOCKET_LOGIN</td>
<td>Error in connecting to the M Login process. Either the M Login process is not running, or the IP address or port number is incorrect.</td>
</tr>
<tr>
<td>MSM_NOSUCHUCI</td>
<td>Either an invalid UCI is specified, or a volume group is specified and the UCI is a NULL pointer.</td>
</tr>
<tr>
<td>MSM_NOSUCHVOLGRP</td>
<td>An invalid volume group is specified.</td>
</tr>
<tr>
<td>MSM_PASSWD</td>
<td>Incorrect password to log into the specified server.</td>
</tr>
<tr>
<td>MSM_SOCKET_SERVER</td>
<td>Unable to connect to the dedicated M server started by the Login process. Verify that free MSM partitions are available.</td>
</tr>
<tr>
<td>MSM_NOPART</td>
<td>Unable to start an M server process.</td>
</tr>
<tr>
<td>MSM_NOPORT</td>
<td>Unable to locate a free dedicated server port.</td>
</tr>
</tbody>
</table>

Example

```c
#include "msmapi.h"
unsigned int sd;
char server[] = "145.123.65.1";
int port = 1666;
char passwd[] = "PEANUTBUTTER";
char uci[] = "MGR";
char volgrp[] = "IBM"
int timeout = 30;
int partsize = 128;
char username[] = "DEFAULT";
sd = msm_login2(server, port, uci, volgrp, passwd, timeout, partsize, username);
if (sd > MSM_BASE_ERROR)
{
    apierror("msm_login", sd);
}
```
**msm_logout()**

Terminates the connection to an MSM system.

**Syntax**

```c
int msm_logout (  
    int sd );
```

**Parameter** | **Description**  
--- | ---  
`sd` | A descriptor that identifies the MSM system with which this function communicates.

**Description**

Use this function call to terminate the connection to the server that is identified by the system descriptor `sd`. If the system descriptor identifies a remote server, the M server process also terminates. If this function is not called to terminate the connection and a timeout is not specified for the server, the function continues to exist in the MSM environment after the client application terminates.

**Return values**

If the function succeeds, it returns MSM_SUCCESS. If the function is not successful, an error code is returned. The most common error codes for this function are listed below. In some situations, other error codes are returned. Refer to “Return Codes” in this manual for a full error code list.

**Code** | **Description**  
--- | ---  
MSM_SDERR | An invalid system descriptor was passed.  
MSM_NETDOWN | The Windows socket implementation detects that the network subsystem failed. If the M server does not terminate, use the M routine APIMGR to halt it.

**Example**

```c
#include "msmapi.h"
int return_code;
int sd;
if ((return_code = msm_logout(sd)) != MSM_SUCCESS)
    apierror("msm_logout", return_code);
```
msm_options()

Modifies the behavior of MSM-Activate/API.

Syntax

```c
int msm_options ( 
    unsigned int setbits, 
    unsigned int clearbits );
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setbits</td>
<td>Option bits to set.</td>
</tr>
<tr>
<td>Clearbits</td>
<td>Option bits to clear.</td>
</tr>
</tbody>
</table>

Description

This function modifies option bits that control MSM-Activate/API’s behavior. If this function is needed, call it after `msm_initialize()`.

The following option bits are defined:

**Option Bits**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
</table>
| 0   | If set, the 16-bit MSM-Activate/API blocks events during its TCP/IP socket operations, such as the period that may elapse between submitting an XECUTE and receiving the response. Although blocking prevents problems that can arise because MSM-Activate/API is not reentrant, the keyboard, mouse, and timer events of all other 16-bit applications on the client also are blocked.

If clear, the 16-bit MSM-Activate/API does not block events. The MSM_BLOCKING error code is returned if a reentrant MSM-Activate/API call is attempted.

This bit defaults to set. In a 32-bit client, bit 0 has no significance. Although events for the client process always are blocked during socket operations, this bit does not affect events for other 32-bit applications or for 16-bit applications.

| 1   | If set, the MSM-Activate/API performs no retries during login with `msm_login`. If clear, the MSM-Activate/API attempts the TCP/IP connection three times before failing.

This bit defaults to clear.

For compatibility with OPEN/USE, `clearbits` are applied before `setbits`.

Return codes

Always returns `MSM_SUCCESS`.

Example

```c
#include "msmapi.h"

/* log into an MSM-Activate/API system */
msm_options(2,1)  /* clear bit 0 and set bit 1 */
```
msm_partsize()

Modifies the partition size of the dedicated server.

Syntax

```c
int msm_partsize (
    unsigned int sd,
    int partsize );
```

Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sd</td>
<td>A descriptor that identifies the connected MSM system with which this function communicates.</td>
</tr>
<tr>
<td>partsize</td>
<td>The new partition size (in kilobytes).</td>
</tr>
</tbody>
</table>

Description

This function modifies the partition size of a running MSM-Activate/API M server. Call this function if MSM_PGMOV errors are returned from other MSM-Activate/API function calls. The new partition size must be greater than or equal to 20 and less than or equal to MSMK_MAXPSIZE.

Return codes

If the function succeeds, it returns MSM_SUCCESS. If the function is unsuccessful, an error code is returned. The most common error codes for this function are listed below. In some situations, other error codes are returned. Refer to “Return Codes” in this manual for a full error code list.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_INVPARTSIZE</td>
<td>The requested partition size is invalid. The partition size must be in the range 20 &lt;= Partition Size &lt;= MSMK_MAXPSIZE.</td>
</tr>
<tr>
<td>MSM_PARTSIZE</td>
<td>The MSM system does not allocate the new partition size.</td>
</tr>
</tbody>
</table>

Example

```c
#include "msmapi.h"
int return_code;
int partsize = 128;
unsigned int sd;

/* log into an MSM-Activate/API system */
.
if ((return_code = msm_partsize(sd, partsize)) != MSM_SUCCESS)
    apierror("msm_partsize", return_code);
```
**msm_rundown()**

Runs down the MSM-Activate/API user interface.

**Syntax**

```c
int msm_rundown (void);
```

**Description**

Use this function to run down the MSM-Activate/API client interface. This function call deallocates the WinSock resources, if used, and terminates the M server processes that are running in the local MSM system. Any open connections to MSM-Server systems are terminated. If additional remote connections are needed after calling this function, recall `msm_initialize()`.

**Return values**

If the function succeeds, it returns `MSM_SUCCESS`. If the function is unsuccessful, an error code is returned. The most common error codes for this function are listed below. In some situations, other error codes are returned. Refer to “Return Codes” in this manual for a full error code list.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_NETINIT</td>
<td>The network subsystem was not initialized with <code>msm_initialize()</code>.</td>
</tr>
<tr>
<td>MSM_NETDOWN</td>
<td>The Windows socket implementation detects that the network subsystem failed.</td>
</tr>
</tbody>
</table>

**Example**

```c
#include "msmapi.h"
int return_code;
if ( ((return_code = msm_rundown()) != MSM_SUCCESS) )
  apierror("msm_rundown", return_code);
```
**msm_sdb_clear()**

Sets the current subscript count in a subscript data block to zero.

**Syntax**

```c
int msm_sdb_clear (  
    struct subscript_data_block *sdb);  
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sdb</td>
<td>A pointer to the SDB that is modified by this function.</td>
</tr>
</tbody>
</table>

**Description**

This function sets the SDB subscript count to zero in the SDB. When passed to the global manipulation function, the global specified in the SDB structure references the top node of the global.

**Return values**

If the function succeeds, it returns MSM_SUCCESS. If the function is unsuccessful, an error code is returned. The most common error codes for this function are listed below. In some situations, other error codes are returned. Refer to “Return Codes” in this manual for a full error code list.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_SDB</td>
<td>The argument is not a valid SDB.</td>
</tr>
</tbody>
</table>

**Example**

```c
#include "msmapi.h"

int return_code;
struct subscript_data_block sdb;

.  // Login and populate structures

if (return_code = msm_sdb_clear(&sdb))
    apierror("msm_sdb_clear", return_code);
```
**msm_sdb_count()**

Returns the current subscript count from a subscript data block.

**Syntax**

```c
int msm_sdb_count(
    struct subscript_data_block *sdb,
    int *count);
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sdb</td>
<td>A pointer to the SDB that is modified by this function.</td>
</tr>
<tr>
<td>count</td>
<td>A pointer to the variable that receives a count of the number of subscripts.</td>
</tr>
</tbody>
</table>

**Description**

This function returns the current subscript count in the SDB to `count`.

**Return values**

If the function succeeds, it returns MSM_SUCCESS. If the function is not successful, an error code is returned. The most common error codes for this function are listed below. In some situations, other error codes are returned. Refer to “Return Codes” in this manual for a full error code list.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_SDB</td>
<td>The argument is not a valid SDB.</td>
</tr>
</tbody>
</table>

**Example**

```c
#include "msmapi.h"
int return_code;
struct subscript_data_block sdb;
int *count;
.
. Login and populate structures
.
if (return_code = msm_sdb_count(&sdb, count))
    apierror("msm_sdb_count", return_code);
```
**msm_sdb_extract()**

Copies the value of a single subscript from a subscript data block to a string.

**Syntax**

```c
msm_sdb_extract(
    struct subscript_data_block  *sdb,
    int  position,
    TCHAR  *value,
    int  bufsize);
```  

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sdb</td>
<td>A pointer to the SDB that is used by this function.</td>
</tr>
<tr>
<td>position</td>
<td>Specifies an integer that contains the position of the subscript to be retrieved from sdb.</td>
</tr>
<tr>
<td>value</td>
<td>A pointer to the buffer that receives the value of the subscript.</td>
</tr>
<tr>
<td>bufsize</td>
<td>An integer value that indicates the size of the buffer allocated to the value parameter.</td>
</tr>
</tbody>
</table>

**Description**

This function returns the value of the specified subscript `position` from `sdb` into `value`.

**Return values**

If the function succeeds, it returns MSM_SUCCESS. If the function is not successful, an error code is returned. The most common error codes for this function are listed below. In some situations, other error codes are returned. Refer to “Return Codes” in this manual for a full error code list.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_SDB</td>
<td>The argument is not a valid SDB.</td>
</tr>
<tr>
<td>MSM_SUBPOS</td>
<td>An invalid subscript position in an SDB.</td>
</tr>
</tbody>
</table>

**Example**

```c
#include "msmapi.h"
int return_code;
struct subscript_data_block sdb;
int position = 1;
char value[30];
.
. populate structures
.
if (return_code = msm_sdb_extract(&sdb, position, value, 30))
    apierror("msm_sdb_extract",return_code);
```
**msm_sdb_initialize()**

Initializes the elements in a subscript data block.

**Syntax**

```c
int msm_sdb_initialize (  
    int sdb_size,  
    struct subscript_data_block *sdb);
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sdb_size</td>
<td>Specifies the size of the SDB that is allocated by the calling function. This should be passed as the constant MSMK_SDB_SIZE.</td>
</tr>
<tr>
<td>sdb</td>
<td>A pointer to the SDB that is used by this function.</td>
</tr>
</tbody>
</table>

**Description**

This function must be called to initialize an SDB prior to use. The size of the SDB is assigned to the first element of the SDB structure, the current subscript count (second element) is set to zero, and the internal elements are initialized for use by MSM.

**Return values**

If the function succeeds, it returns MSM_SUCCESS. If the function is not successful, an error code is returned. The most common error codes for this function are listed below. In some situations, other error codes are returned. Refer to “Return Codes” in this manual for a full error code list.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_SDB</td>
<td>The sdb parameter is not a valid SDB.</td>
</tr>
</tbody>
</table>

**Example**

```c
#include "msmapi.h"
int return_code;
int sdb_size=MSMK_SDB_SIZE;
struct subscript_data_block sdb;
if (return_code = msm_sdb_initialize(sdb_size, &sdb))  
    apierror("msm_sdb_initialize", return_code);
```
**msm_sdb_insert()**

Inserts a subscript into a subscript data block.

**Syntax**

```c
int msm_sdb_insert(
    int position,
    TCHAR *value,
    struct subscript_data_block *sdb);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>Specifies the position of the subscript to be inserted.</td>
</tr>
<tr>
<td>value</td>
<td>A pointer to the character string that contains the subscript to be inserted.</td>
</tr>
<tr>
<td>sdb</td>
<td>A pointer to the SDB that is used by this function.</td>
</tr>
</tbody>
</table>

**Description**

The string specified in `value` is inserted into the SDB at the location specified by `position`. The current subscript count is updated with the value of `position`. If the current subscript count in `sdb` is greater than `position`, the count is set to `position` and any later subscripts are cleared. Insert subscripts into the SDB in ascending order. The value of `position` cannot be more than the current subscript count plus one.

**Return values**

If the function succeeds, it returns MSM_SUCCESS. If the function is unsuccessful, an error code is returned. The most common error codes for this function are listed below. In some situations, other error codes are returned. Refer to “Return Codes” in this manual for a full error code list.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_SDB</td>
<td>The argument is not a valid SDB.</td>
</tr>
<tr>
<td>MSM_SUBPOS</td>
<td>An invalid subscript position in an SDB.</td>
</tr>
<tr>
<td>MSM_INVSUBSCR</td>
<td>An invalid subscript. The total subscript length is greater than the maximum allowed by MSM.</td>
</tr>
</tbody>
</table>

**Example**

```c
#include "msmapi.h"
int return_code;
struct subscript_data_block sdb;
int position = 1;
char value[] = "SUB1";
if (return_code = msm_sdb_insert(position, value, &sdb))
    apierror("msm_sdb_insert", return_code);
```
msm_sdb_insert_null()

Copies an M empty string to a subscript data block.

Syntax

```c
int msm_sdb_insert_null(  
    int position,  
    struct subscript_data_block *sdb );
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>Specifies the position of the subscript to be modified.</td>
</tr>
<tr>
<td>sdb</td>
<td>A pointer to the SDB that is used by this function.</td>
</tr>
</tbody>
</table>

Description

This function inserts an M empty string into the subscript identified by the variable `position`. An empty subscript is only valid in the rightmost subscript position when calling `msm_global_order`. Insert subscripts into the SDB in ascending order. If the current subscript count in the SDB is greater than `position`, the count is set to `position` and any later subscripts are cleared. The value of `position` cannot be more than the current subscript count plus one.

Return values

If the function succeeds, it returns MSM_SUCCESS. If the function is not successful, an error code is returned. The most common error codes for this function are listed below. In some situations, other error codes are returned. Refer to “Return Codes” in this manual for a full error code list.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_SDB</td>
<td>The argument is not a valid SDB.</td>
</tr>
<tr>
<td>MSM_SUBPOS</td>
<td>An invalid subscript position in an SDB.</td>
</tr>
<tr>
<td>MSM_INVSUBSCR</td>
<td>An invalid subscript. The total subscript length is greater than the maximum allowed by MSM.</td>
</tr>
</tbody>
</table>

Example

```c
#include "msmapi.h"
int return_code;
struct subscript_data_block sdb;
int position = 1;

// Login and populate structures & arrays

if (return_code = msm_sdb_insert_null(position, &sdb))
    apierror("msm_sdb_insert_null", return_code);
```
**msm_timeout()**

Modifies the client-side timeout.

**Syntax**

```c
int msm_timeout (  
    unsigned int  sd,  
    int  timeout );
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sd</td>
<td>A descriptor that identifies the connected MSM system with which this function communicates.</td>
</tr>
<tr>
<td>timeout</td>
<td>The new client timeout (in seconds).</td>
</tr>
</tbody>
</table>

**Description**

This function modifies the client-side timeout value, which is the amount of time the MSM-Activate/API waits for a response from the server before it concludes that the server job crashed. Increase this value if the command you are executing on the server requires a long time to complete (such as a large database search). The MSM-Activate/API then waits for the command to complete rather than assumes that it failed.

**Note** The parameter `timeout` that is specified with any of the `msm_login` functions is not related to this timeout. When this timeout elapses, it is assumed that the server has crashed. When the timeout on the login functions elapses, the server assumes that its services are no longer needed.

**Return codes**

If the function succeeds, it returns MSM_SUCCESS. If the function is unsuccessful, an error code is returned. The most common error codes for this function are listed below. In some situations, other error codes are returned. Refer to “Return Codes” in this manual for a full error code list.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_TIMEOUT</td>
<td>Attempt to set a timeout of &lt;0.</td>
</tr>
<tr>
<td>MSM_SDERR</td>
<td>Socket descriptor not found.</td>
</tr>
</tbody>
</table>

**Example**

```c
#include "msmapi.h"

int return_code;  
int timeout = 128;  
unsigned int sd;  

/* log into an MSM-Activate/API system */  

if ((return_code = msm_timeout(sd, timeout)) != MSM_SUCCESS)  
    apierror("msm_timeout", return_code);
```
**msm_version()**

Returns the MSM-Activate/API version number.

**Syntax**

```c
int msm_version ( 
    TCHAR *vers );
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vers</td>
<td>Specifies a pointer to a character string to receive the version information.</td>
</tr>
</tbody>
</table>

**Description**

This function obtains a string that identifies the version number of the MSM-Activate/API client.

The following version numbers also may be relevant in this context:

- The version number of the MSM-Server.
- The version number of the portion of MSM-Activate/API that resides on the MSM-Server.

**Return codes**

Always returns MSM_SUCCESS.

**Example**

```c
#include "msmapi.h"
char vers[MSMK_MAXRETURNLEN]

/* log into an MSM-Activate/API system */
msm_version(vers)
```
msm_xecute()  

Executes a line of MSM code using the MSM XECUTE command.

Syntax

```c
int msm_xecute (
    int sd, 
    TCHAR *mcode, 
    TCHAR *result, 
    int bufsize);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sd</td>
<td>A descriptor that identifies the MSM system with which this function communicates.</td>
</tr>
<tr>
<td>mcode</td>
<td>The M code to execute.</td>
</tr>
<tr>
<td>result</td>
<td>A buffer that holds the result of the execution.</td>
</tr>
<tr>
<td>bufsize</td>
<td>An integer value that indicates the size of the buffer allocated to the <code>result</code> parameter.</td>
</tr>
</tbody>
</table>

Description

This function executes the M code that is held in `mcode` and returns the result into `result`. Sufficient memory must be allocated in the `result` parameter for the returned data.

This function should not be used to run an interactive MSM routine. Use it to call functions, routines, or M statements that return data or an empty string.

**Note**  The returned data is truncated if the allocated buffer (`*result`) is too small to accept it.

Return values

If the function succeeds, it returns MSM_SUCCESS. If the function is not successful, an error code is returned. Because this function can execute any M code, any MSM error can be returned. If an error occurs, the full M error is returned in `value`. Refer to “Return Codes” in this manual for a full error code list.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_INCOMPLETEDATA</td>
<td>The data returned from this call is larger than the space allocated for the return value parameter.</td>
</tr>
</tbody>
</table>
C Example

```c
#include "msmapi.h"
int return_code;
unsigned int sd;
char mcode[100];
char result[MSMK_MAXRETURNLEN];
strcpy(mcode, "S A=$$UPDATE^DBASE(10) WRITE A");
if (return_code = msm_xecute(sd, mcode, result, MSMK_MAXRETURNLEN))
    apierror("msm_xecute",return_code);
```

Visual Basic Example

```vb
retsize% = 512
result$  = SPACE(retsize)
mcode$   = "S A=$$UPDATE^DBASE(10) WRITE A")
return_code = msm_xecute(sd, mcode$, result$, retsize%)
if return_code <> 0
    apierror("msm_xecute", return_code)
End If
```
** msm_xecute2()**

Executes a line of MSM code using the MSM XECUTE command.

**Syntax**

```c
int msm_xecute2 (  
    int sd,  
    TCHAR *mcode,  
    TCHAR *result,  
    int *bufsize );
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sd</td>
<td>A descriptor that identifies the MSM system with which this function communicates.</td>
</tr>
<tr>
<td>mcode</td>
<td>M code to execute.</td>
</tr>
<tr>
<td>result</td>
<td>A buffer that holds the result of the execution.</td>
</tr>
<tr>
<td>bufsize</td>
<td>The address of an integer value that indicates the size of the buffer allocated to the result parameter. On return, this value will contain the actual size of the data returned in result.</td>
</tr>
</tbody>
</table>

**Description**

This function executes the M code that is held in mcode and returns the result into result. Sufficient memory must be allocated in the result parameter for the returned data. The bufsize variable is returned with the length of the data received.

This function should not be used to run an interactive MSM routine. Use it to call functions, routines, or M statements that return data or an empty string.

**Note**  The returned data is truncated if the allocated buffer (*result) is too small to accept it.

**Return values**

If the function succeeds, it returns MSM_SUCCESS. If the function is not successful, an error code is returned. Because this function can execute any M code, any MSM error can be returned. If an error occurs, the full M error is returned in value. Refer to “Return Codes” in this manual for a full error code list.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_INCOMPLETEDATA</td>
<td>The data returned from this call is larger than the space allocated for the return value parameter.</td>
</tr>
</tbody>
</table>
C Example

```c
#include "msmapi.h"
int return_code;
int size;
unsigned int sd;
char mcode[100];
char result[MSMK_MAXRETURNLEN];
size = MSMK_MAXRETURNLEN;
strcpy(mcode, "S A-$$UPDATE^DBASE(10) WRITE A");
if (return_code = msm_xecute2(sd, mcode, result, &size))
    apierror("msm_xecute", return_code);
```

Visual Basic Example

```vb
retsize% = 512
result$  = SPACE(retsize)
mcode$   = "S A-$$UPDATE^DBASE(10) WRITE A")
return_code = msm_xecute2(sd, mcode$, result$, retsize%)
if return_code <> 0
    apierror("msm_xecute", return_code)
End If
```
Overview

MSM-Activate/Java enables a Java client to connect to an MSM-Activate server, where it can run M code, store and retrieve M data, and execute M commands. The techniques used are similar to those supported by MSM-Activate/API, with Java method names matching API library function names.

Each object instantiated from the principal class, MSMAPI, is capable of making a single connection via TCP/IP to an MSM-Activate server. Four other supporting classes enable parameters to be passed by reference to methods of the principal class. The classes are thread-safe, permitting different execution threads to use a single object instance safely.

A Bean variant is provided in a file called MSMAPIJAR for use within Java development environments that support Beans. The JAR file’s classes are identical to those in the five separate class-files.

Compatibility with previous version

MSM-API Version 4.0, the predecessor of MSM-Activate, included Java components. When you update to MSM-Activate, the following product change may require you to modify existing applications that use the Java component.

Classes renamed

Because some platforms are restricted to 8.3 file names, the names of the MSM-Activate class-files (before the extension) do not exceed eight characters. On an 8.3 system, the extension becomes .CLA instead of .CLASS although the basic file name remains unchanged. This has resulted in the following changes to class names:

- DO2PARAMS becomes DO2PARMS
- StringRef becomes StrRef
MSMAPI Class and Methods

The principal class is called MSMAPI. An object instantiated from this class is capable of making a single connection via TCP/IP to an MSM-Activate server. It implements the following methods, all of which return a result code. Refer to “Result Codes” in this manual for additional information.

**msm_login Method**

**Syntax**

```java
int msm_login(
    StrRef version)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>Identifies the version of MSM-Activate server to which a connection has been established.</td>
</tr>
</tbody>
</table>

**Description**

This method is primarily used with the Bean, where properties governing the login are set prior to the method being called.

Upon successful return, the parameter `version` contains a string identifying the version of MSM-Activate server to which a connection has been established. When the server is Unicode-capable, the string has a ‘/’ suffix.
**msm_login2 Method**

**Syntax**

```java
int msm_login2(
    String server,
    int port,
    String uci,
    String vg,
    String password,
    int timeout,
    int partsize,
    String username )
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>Specifies the MSM-Activate server’s hostname or IP address.</td>
</tr>
<tr>
<td>port</td>
<td>Specifies the MSM-Activate server’s port.</td>
</tr>
<tr>
<td>uci</td>
<td>Specifies which UCI of the server is to receive the connection.</td>
</tr>
<tr>
<td>vg</td>
<td>Specifies which VG of the server is to receive the connection.</td>
</tr>
<tr>
<td>password</td>
<td>If the server’s security has been configured, submits a suitable password.</td>
</tr>
<tr>
<td>timeout</td>
<td>Sets the number of seconds of connection inactivity that will cause the MSM-Activate server to close the connection.</td>
</tr>
<tr>
<td>partsize</td>
<td>Dictates the size (in kilobytes) of the M partition to be obtained on the server. Minimum value is 20 and maximum is 256. Zero denotes the server’s default partition size or 64, whichever is larger.</td>
</tr>
<tr>
<td>username</td>
<td>Works in conjunction with the password to validate the login request against the server’s MSM-Activate security settings.</td>
</tr>
</tbody>
</table>

**Description**

Use this method to log in to the MSM-Activate server specified by `server` and `port`, and establish a connection with the UCI and Volume Group specified by `uci` and `vg`.

The `timeout` parameter sets the number of seconds of connection inactivity that will cause the MSM-Activate server to close the connection. Use the `msm_keepalive` method to reset the timer and keep a connection open. A value of zero denotes that no server timeout is to occur.

The `username` parameter works in conjunction with `password` to validate the login request against the server’s MSM-Activate security settings. Neither string is encrypted before transmission over the network.
**msm_login3 Method**

This method is an extension of msm_login2, which returns version information from the server.

**Syntax**

```java
int msm_login3(
    String server,
    int port,
    String uci,
    String vg,
    String password,
    int timeout,
    int partsize,
    String username,
    StrRef version )
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>Specifies the MSM-Activate server’s hostname or IP address.</td>
</tr>
<tr>
<td>port</td>
<td>Specifies the MSM-Activate server’s port.</td>
</tr>
<tr>
<td>uci</td>
<td>Specifies which UCI of the server is to receive the connection.</td>
</tr>
<tr>
<td>vg</td>
<td>Specifies which VG of the server is to receive the connection.</td>
</tr>
<tr>
<td>password</td>
<td>If the server’s security has been configured, submits a suitable password.</td>
</tr>
<tr>
<td>timeout</td>
<td>Sets the number of seconds of connection inactivity that will cause the MSM-Activate server to close the connection.</td>
</tr>
<tr>
<td>partsize</td>
<td>Dictates the size (in kilobytes) of the M partition to be obtained on the server. Minimum value is 20 and maximum is 256. Zero denotes the server’s default partition size or 64, whichever is larger.</td>
</tr>
<tr>
<td>username</td>
<td>Works in conjunction with password to validate the login request against the server’s MSM-Activate security settings.</td>
</tr>
<tr>
<td>version</td>
<td>Identifies the version of MSM-Activate server to which a connection has been established.</td>
</tr>
</tbody>
</table>

**Description**

Use this method to log in to the MSM-Activate server specified by `server` and `port`, and establish a connection with the UCI and Volume Group specified by `uci` and `vg`.

The `timeout` parameter sets the number of seconds of connection inactivity that will cause the MSM-Activate server to close the connection. Use the `msm_keepalive` method to reset the timer and keep a connection open. A value of zero denotes no server timeout is to occur.

The `username` parameter works in conjunction with `password` to validate the login request against the server’s MSM-Activate security settings. Neither string is encrypted before transmission over the network.

Upon successful return the parameter `version` contains a string identifying the version of MSM-Activate server to which a connection has been established. When the server is Unicode-capable, the string will have a '/U' suffix.
msm_do Method

Syntax

int msm_do(
    String msmcode,
    String msmparms,
    CharRef ret)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msmcode</td>
<td>Specifies the label and routine of the M function or subroutine to be called.</td>
</tr>
<tr>
<td>msmparms</td>
<td>Contains a comma-separated list of the arguments to be submitted to the M function or subroutine.</td>
</tr>
<tr>
<td>ref</td>
<td>Provides the return value when an M function is called.</td>
</tr>
</tbody>
</table>

Description

This method is provided to assist migration from legacy MSM-API applications using the msm_do function. For new development, the msm_do2 method is preferable.

The M function or subroutine specified by the msmcode parameter is called, for example: AddRec^FILER. The arguments in msmparms are submitted to the function or subroutine.

Where an M function is called its return value is provided in ret. If the method’s return value is greater than MSMAPI.MSM_M_ERROR, then ret will contain additional information about the M error that occurred.

MSM-Activate does not encrypt its transmissions over the network.

msm_do2 Method

Syntax

int msm_do2(
    String msmcode,
    DO2PARMS msmparms,
    CharRef ret)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msmcode</td>
<td>Specifies the label and routine of the M function or subroutine to be called.</td>
</tr>
<tr>
<td>msmparms</td>
<td>Contains an instance of the DO2PARMS supporting class, whose member variables contain the arguments to be submitted to the M function or subroutine.</td>
</tr>
<tr>
<td>ref</td>
<td>Provides the return value when an M function is called.</td>
</tr>
</tbody>
</table>
Description

This method calls the M function or subroutine specified by the msmcode parameter, for example: AddRec^FILER. The arguments in msmparams are submitted to the function or subroutine.

The msmparams parameter is an instance of the DO2PARMS supporting class whose member variables contain the arguments to be submitted to the M function or subroutine. For additional information, refer to the documentation for the DO2PARMS class.

When an M function is called, its return value is provided in ret. If the method’s return value is greater than MSMAPI.MSM_M_ERROR, then ret will contain additional information about the M error that occurred.

MSM-Activate does not encrypt its transmissions over the network.

**msm_xecute Method**

**Syntax**

```java
int msm_xecute(
    String xecute,
    CharRef ret)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xecute</td>
<td>Specifies M code to be executed on the server.</td>
</tr>
<tr>
<td>ref</td>
<td>Provides the output from the execution of xecute.</td>
</tr>
</tbody>
</table>

**Description**

Use this method to execute a string of M code. The xecute parameter specifies M code to be Xecuted on the server, and upon successful return from the method ret, it will contain the output of this execution.

If the method’s return value is greater than MSMAPI.MSM_M_ERROR, then ret will contain additional information about the M error that occurred.

MSM-Activate does not encrypt its transmissions over the network.
msm_logout Method

Syntax
int msm_logout()

Description
This method attempts to close the object’s open connection. If no connection is currently established, it returns the MSMAPI.MSM_CONNECTION error code.

If a previous msm_do, msm_do2, or msm_xecute method failed with an error that indicates a connection problem (for example: the server-side timeout may have expired), then it is still necessary to call this method before attempting reconnection.

msm_version Method

Syntax
int msm_version(
    StrRef str
)

Parameter Description
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>Identifies the version of MSM-Activate/Java being used.</td>
</tr>
</tbody>
</table>

Description
This method returns in str the version of MSM-Activate/Java being used. Use this method to determine that you have the client-side support you require.

msm_keepalive Method

Syntax
msm_keepalive()

Description
Use this method to reset the server-side timeout counter for the connection, in order to prevent the connection from being closed by the server because of client inactivity. The timeout interval is a parameter of the msm_login2 and msm_login3 methods. If no connection is currently established, then msm_keepalive returns the MSMAPI.MSM_CONNECTION error code.
Supporting Classes

MSM-Activate’s low-level interfaces use pointers to permit by-reference parameter passing and to return results. Since the Java language does not include pointers, the following supporting classes are used to provide equivalent functionality. The Java developer creates an instance of the appropriate class, sets its member variable(s) if required, and passes the instance as a parameter of an MSMAPI method. On return from the method, the instance’s member variable(s) can be queried.

CharRef Class
This class contains a single member variable called \textit{value} which is an array of \textit{chars}.

DO2PARMS Class
This class is used in conjunction with the msm_do2 method. It contains the following member variables:

\textbf{nParams}
This variable is of type \textit{int} and must be set to the number of parameters the object represents.

\textbf{flag}
This variable is an array of type \textit{boolean}. It must contain the number of elements specified by \textit{nParams}. A value of \textit{TRUE} indicates that the corresponding parameter is passed by reference to the M subroutine or function, receiving a value on return from the method. A value of \textit{FALSE} indicates that the corresponding parameter is passed by value to the M subroutine or function and is unchanged on return from the method.

\textbf{params}
This variable is an array of type \textit{CharRef}. It must contain the number of elements specified by \textit{nParams}. Where a value is to be passed into the M subroutine or function in the corresponding parameter, set the value into the matching element before the method is invoked. For parameters given a \textit{flag} value \textit{TRUE}, check the matching element on return.

IntRef Class
This class contains a single member variable called \textit{value} which is of type \textit{int}.

StrRef Class
This class contains a single member variable called \textit{value} which is of type \textit{String}.
The Bean

MSM-Activate/Bean presents MSM-Activate as a component that can be used in development environments that support the JavaBeans architecture. One example of such an environment is Symantec’s Visual Café.

Methods

Since the bean is an alternative form of MSM-Activate/Java, it supports the same methods.

Properties

The following properties are available to the bean.

In accordance with JavaBean’s architecture, each property is implemented as a get/set method pair. An applet also can access the properties. For example, the getIsUnicode() method returns the IsUnicode property and setIsUnicode(value) sets it.

IsUnicode Property

boolean

If set to TRUE before a login method, the login will fail unless the MSM-Activate server is Unicode-capable. If the property is FALSE (the default) when a login method is invoked, then after the method succeeds, the property will be set to TRUE if the server is Unicode-capable.

The msm_logout method restores the property to the value it had before login.

Partsize Property

int

The size (in kilobytes) of the M partition required on the server. A value of zero (the default) denotes that the MSM-Activate server’s default partition size (minimum 64) is to be used. Otherwise, the minimum value is 20 and the maximum is MSMAPI.MSMK_MAXPSIZE (currently 256).

Password Property

String

This property may be required by the MSM-Activate server’s security settings in order to permit login. The Default is “Default.” The MSM-Activate protocol does not encrypt this string before transmitting it across the network.

Port Property

int

The TCP/IP port on which the MSM-Activate server accepts connections. The default is 1666.
**Server Property**

**String**
The IP address or hostname of the system on which the MSM-Activate server is running. The default is 127.0.0.1.

**Timeout Property**

**int**
The number of seconds of inactivity after which the MSM-Activate server will terminate the connection. A value of zero disables the inactivity timeout feature. The default is 30.

**UCI Property**

**String**
The name of the server UCI to which the client connects. An empty string (the default) in this and the VolGrp property will cause the client to access the UCI in which the MSM-Activate server is running.

**Username Property**

**String**
This property may be required by the MSM-Activate server’s security settings in order to permit login. It governs the permissions granted by the server. The default is “Default.” The MSM-Activate protocol does not encrypt this string before transmitting it across the network.

**VolGrp Property**

**String**
The name of the server Volume Group to which the client connects. An empty string (the default) in this property and the UCI property will cause the client to access the UCI in which the MSM-Activate server is running.
Security

As with other variants of MSM-Activate, data transmitted between client and server is not encrypted unless lower levels of the TCP/IP protocol stack do this automatically. Take particular account of this when selecting usernames and passwords to be transmitted by your applet.

Browsers that can host Java applets as components of HTML pages typically enforce certain security restrictions that can impact the ability of MSM-Activate/Java to communicate with an MSM-Activate server. Generally, an applet is permitted network access only to the IP address from which the applet was loaded. Unless the MSM-Activate server runs on the same machine as the webserver (thus sharing the same IP address), the applet may fail to connect to its server. In an intranet context, it may be feasible to arrange for browsers to treat MSMAPI and its supporting classes as trusted classes by placing them on the classpath. Some browsers also are able to grant additional rights to classes that are loaded from a signed file, such as a .CAB or .JAR file.

A browser’s security restrictions may be different when viewing an HTML page loaded from a local file. This might occur during development and testing. For example, Netscape Navigator 4 permits a local file page’s applet to connect to 127.0.0.1 using an untrusted class, but Microsoft’s Internet Explorer (3 or 4) does not. One workaround is to place the Activate classes on the classpath of the developer’s system. Another is to use an applet viewer tool to test the applet. Alternatively, test your pages by loading them from a webserver.

For additional information, refer to “Security Restrictions” in the Line4 sample program.

Samples

The following sample programs are provided with MSM-Activate/Java.

<table>
<thead>
<tr>
<th>Client</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>Test</td>
<td>Illustrates simple connectivity between a Java class and an MSM-Server using MSM-Activate/Java.</td>
</tr>
<tr>
<td>Java</td>
<td>Sample</td>
<td>Illustrates simple connectivity between a Java application in a web browser and an MSM-Server using MSM-Activate/Java.</td>
</tr>
<tr>
<td>Java</td>
<td>Line4</td>
<td>Demonstrates Java-driven communication with an MSM-Server to play a game at various levels of sophistication.</td>
</tr>
</tbody>
</table>

In this manual, refer to “Sample Programs” for the location of installed samples, and to “Setup Information” for general information on setting up servers and clients.
Result Codes

Error reporting is consistent with other variations of MSM-Activate. All the methods return an integer result code. These codes are defined constants of the MSMAPI class. Success is indicated by MSM_SUCCESS, which is defined as 0. Failure values start above MSM_BASE_ERROR, defined as 10000.

The socket implementation provided by the Java language currently provides little information about the cause of any network problem. The MSM_SOCKET_ERROR returned by MSMAPI methods might indicate a client timeout, a server-side disconnect or some other network problem.

The following defined constants are used to indicate errors in the Java variant of MSM-Activate. Refer to “Return Codes” in this manual for additional information.

MSM-Activate/Java Result Code Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_SUCCESS</td>
<td>0</td>
</tr>
<tr>
<td>MSM_BASE_ERROR</td>
<td>10000</td>
</tr>
<tr>
<td>MSM_SOCKET_LOGIN</td>
<td>MSM_BASE_ERROR+1</td>
</tr>
<tr>
<td>MSM_CONNERR</td>
<td>MSM_BASE_ERROR+6</td>
</tr>
<tr>
<td>MSM_INVPARTSIZE</td>
<td>MSM_BASE_ERROR+8</td>
</tr>
<tr>
<td>MSM_PASSWD</td>
<td>MSM_BASE_ERROR+9</td>
</tr>
<tr>
<td>MSM_NOSUCHUCI</td>
<td>MSM_BASE_ERROR+30</td>
</tr>
<tr>
<td>MSM_NOSUCHVOLGRP</td>
<td>MSM_BASE_ERROR+31</td>
</tr>
<tr>
<td>MSM_STRLEN</td>
<td>MSM_BASE_ERROR+34</td>
</tr>
<tr>
<td>MSM_UNSPECIFIED</td>
<td>MSM_BASE_ERROR+35</td>
</tr>
<tr>
<td>MSM_RETURNLEN</td>
<td>MSM_BASE_ERROR+37</td>
</tr>
<tr>
<td>MSM_SECURITY</td>
<td>MSM_BASE_ERROR+38</td>
</tr>
<tr>
<td>MSM_PROTOCOL</td>
<td>MSM_BASE_ERROR+39</td>
</tr>
<tr>
<td>MSM_DO2PARAMS</td>
<td>MSM_BASE_ERROR+40</td>
</tr>
<tr>
<td>MSM_CONNECTED</td>
<td>MSM_BASE_ERROR+41</td>
</tr>
<tr>
<td>MSM_SOCKET_ERROR</td>
<td>MSM_BASE_ERROR+42</td>
</tr>
<tr>
<td>MSM_PSIZE</td>
<td>MSM_BASE_ERROR+44</td>
</tr>
<tr>
<td>MSM_VERSN</td>
<td>MSM_BASE_ERROR+45</td>
</tr>
<tr>
<td>MSM_INVRC</td>
<td>MSM_BASE_ERROR+46</td>
</tr>
<tr>
<td>MSM_ALLOW</td>
<td>MSM_BASE_ERROR+47</td>
</tr>
<tr>
<td>MSM_UTFCONV</td>
<td>MSM_BASE_ERROR+48</td>
</tr>
<tr>
<td>MSM_M_ERROR</td>
<td>MSM_BASE_ERROR+49</td>
</tr>
<tr>
<td>MSM_SBSCR</td>
<td>MSM_M_ERROR+1</td>
</tr>
<tr>
<td>MSM_NAKED</td>
<td>MSM_M_ERROR+2</td>
</tr>
<tr>
<td>MSM_PROT</td>
<td>MSM_M_ERROR+3</td>
</tr>
<tr>
<td>MSM_UNDEF</td>
<td>MSM_M_ERROR+4</td>
</tr>
<tr>
<td>MSM_PGMMOV</td>
<td>MSM_M_ERROR+5</td>
</tr>
<tr>
<td>Constant</td>
<td>Definition</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>MSM_STKOV</td>
<td>MSM_M_ERROR+6</td>
</tr>
<tr>
<td>MSM_SYNTX</td>
<td>MSM_M_ERROR+8</td>
</tr>
<tr>
<td>MSM_NOPGM</td>
<td>MSM_M_ERROR+9</td>
</tr>
<tr>
<td>MSM_MXSTR</td>
<td>MSM_M_ERROR+10</td>
</tr>
<tr>
<td>MSM_DKSER</td>
<td>MSM_M_ERROR+11</td>
</tr>
<tr>
<td>MSM_DKHER</td>
<td>MSM_M_ERROR+12</td>
</tr>
<tr>
<td>MSM_CMMND</td>
<td>MSM_M_ERROR+13</td>
</tr>
<tr>
<td>MSM_FUNCT</td>
<td>MSM_M_ERROR+14</td>
</tr>
<tr>
<td>MSM_INDER</td>
<td>MSM_M_ERROR+15</td>
</tr>
<tr>
<td>MSM_DIVER</td>
<td>MSM_M_ERROR+17</td>
</tr>
<tr>
<td>MSM_DKFUL</td>
<td>MSM_M_ERROR+18</td>
</tr>
<tr>
<td>MSM_LINER</td>
<td>MSM_M_ERROR+19</td>
</tr>
<tr>
<td>MSM_MERGE</td>
<td>MSM_M_ERROR+20</td>
</tr>
<tr>
<td>MSM_MODER</td>
<td>MSM_M_ERROR+21</td>
</tr>
<tr>
<td>MSM_MXMEM</td>
<td>MSM_M_ERROR+22</td>
</tr>
<tr>
<td>MSM_MXNUM</td>
<td>MSM_M_ERROR+23</td>
</tr>
<tr>
<td>MSM_NODEV</td>
<td>MSM_M_ERROR+24</td>
</tr>
<tr>
<td>MSM_NOPEN</td>
<td>MSM_M_ERROR+25</td>
</tr>
<tr>
<td>MSM_SYSTM</td>
<td>MSM_M_ERROR+26</td>
</tr>
<tr>
<td>MSM_DDPER</td>
<td>MSM_M_ERROR+27</td>
</tr>
<tr>
<td>MSM_BADCH</td>
<td>MSM_M_ERROR+28</td>
</tr>
<tr>
<td>MSMASYNC</td>
<td>MSM_M_ERROR+29</td>
</tr>
<tr>
<td>MSM_BKERR</td>
<td>MSM_M_ERROR+30</td>
</tr>
<tr>
<td>MSM_DKRES</td>
<td>MSM_M_ERROR+31</td>
</tr>
<tr>
<td>MSM_DPARM</td>
<td>MSM_M_ERROR+32</td>
</tr>
<tr>
<td>MSM_EXPER</td>
<td>MSM_M_ERROR+33</td>
</tr>
<tr>
<td>MSM_INHIB</td>
<td>MSM_M_ERROR+34</td>
</tr>
<tr>
<td>MSM_ISYNT</td>
<td>MSM_M_ERROR+35</td>
</tr>
<tr>
<td>MSM_LCNSE</td>
<td>MSM_M_ERROR+36</td>
</tr>
<tr>
<td>MSM_MTERR</td>
<td>MSM_M_ERROR+37</td>
</tr>
<tr>
<td>MSM_NOSYS</td>
<td>MSM_M_ERROR+38</td>
</tr>
<tr>
<td>MSM_NOUCI</td>
<td>MSM_M_ERROR+39</td>
</tr>
<tr>
<td>MSM_PCERR</td>
<td>MSM_M_ERROR+40</td>
</tr>
<tr>
<td>MSM_PLDER</td>
<td>MSM_M_ERROR+41</td>
</tr>
<tr>
<td>MSM_VWERR</td>
<td>MSM_M_ERROR+42</td>
</tr>
<tr>
<td>MSM_ZCALL</td>
<td>MSM_M_ERROR+43</td>
</tr>
<tr>
<td>MSM_ZCERR</td>
<td>MSM_M_ERROR+44</td>
</tr>
<tr>
<td>MSM_ZSAVE</td>
<td>MSM_M_ERROR+45</td>
</tr>
<tr>
<td>MSM_ZSVGP</td>
<td>MSM_M_ERROR+46</td>
</tr>
<tr>
<td>MSM_TPROC</td>
<td>MSM_M_ERROR+47</td>
</tr>
<tr>
<td>MSM_DSCON</td>
<td>MSM_M_ERROR+48</td>
</tr>
<tr>
<td>MSM_CLOBR</td>
<td>MSM_M_ERROR+49</td>
</tr>
</tbody>
</table>
Overview

MSM-Activate is accompanied by sets of sample programs and sample files that demonstrate various features of the MSM-Activate interface. This chapter describes these samples.

Sample Programs

The current version of MSM-Activate is distributed with the sample programs listed in the following table. The files for these demonstrations are stored under the subdirectory Samples of the directory where MSM-Activate is installed (typically C:\Program Files\Micronetics\MSM-Activate). Visual Basic samples are in Samples\VB Demos, C samples are in Samples\C Demo, Java samples are in Samples\Java, and Delphi samples are in Samples\Delphi.

<table>
<thead>
<tr>
<th>Client</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++</td>
<td>TEST</td>
<td>Exercises the low-level interfaces of MSM-Activate/API.</td>
</tr>
<tr>
<td>Delphi</td>
<td>OLEDemo</td>
<td>Illustrates simple connectivity between a Delphi application and an MSM-Server using MSM-Activate/COM.</td>
</tr>
<tr>
<td>Java</td>
<td>Line4</td>
<td>Demonstrates Java-driven communication with an MSM-Server to play a game at various levels of sophistication.</td>
</tr>
<tr>
<td>Java</td>
<td>Sample</td>
<td>Illustrates simple connectivity between a Java application in a web browser and an MSM-Server using MSM-Activate/Java.</td>
</tr>
<tr>
<td>Java</td>
<td>Test</td>
<td>Illustrates simple connectivity between a Java class and an MSM-Server using MSM-Activate/Java.</td>
</tr>
<tr>
<td>Visual Basic</td>
<td>Detail</td>
<td>Illustrates how to use MSM-Activate/API to retrieve information from an MSM database.</td>
</tr>
<tr>
<td>Visual Basic</td>
<td>OLEsimp3</td>
<td>Illustrates how to use MSM-Activate/COM access to an MSM-Server or MSM-Workstation to manipulate local variables in MSM from a program created with Visual Basic.</td>
</tr>
<tr>
<td>Visual Basic</td>
<td>OLEsimp4</td>
<td>Illustrates simple connectivity between a 32-bit Visual Basic application and an MSM-Server using MSM-Activate/API.</td>
</tr>
<tr>
<td>Visual Basic</td>
<td>Sample32</td>
<td>Illustrates how to use MSM-Activate/API to manipulate local variables in MSM from a program created with Visual Basic.</td>
</tr>
</tbody>
</table>
Setup Information

Servers
All of the sample programs demonstrate the interoperability between either an MSM-Server or an MSM-Workstation-built Automation server and an external software product. For these demonstrations to function properly, either the MSM-Server must be up and running and the server-side component of MSM-Activate must be active; or if you are using an MSM-Workstation Automation server, then the relevant instance of the Automation server must be compiled and ready to run.

Refer to “Managing an MSM-Activate Server” in this manual for assistance with starting the server-side component of MSM-Activate.

Clients
On the client side, the following parameters must be specified correctly before a connection can be made with an MSM-Server:

- The IP address of the MSM-Server or the location of the MSM-Workstation-built Automation server. In test environments, it is often possible to use the “internal loop-back” number 127.0.0.1. In a network in which a proxy server is active, it is sometimes necessary to supply the actual IP address of the MSM-Server, even if the server physically resides on the same computer as the client that is used to run the sample.

- The port number that is used by the MSM-Activate server to listen for new requests for connecting to an MSM-Server system. In all of the demonstrations, port number 1666 is used for this purpose. When your system uses a different port number, you must substitute the correct port number.

- The names of the Volume Group and the UCI in which MSM software and databases are to be accessed when connecting to an MSM-Server system. In most of the demonstrations, it is assumed that the UCI “MGR” in volume group 0 may be used. When local security restrictions prohibit such access, these parameters must be modified to reflect the actual situation.

- A username and password that allow access into the MSM-Server according to the MSM-Activate security settings on the server.
TEST (C++)

This sample C++ program exercises the low-level interfaces of MSM-Activate/API.

Background

The TEST sample application includes examples of most function calls that can be made to MSM-Activate/API. The files are in the subdirectory Samples\CDemo of the MSM-Activate directory.

There are three versions of the sample:

- TEST.EXE (16-bit sample for Microsoft Visual C++, Version 1.52)
- TEST32.EXE (32-bit sample for Microsoft Visual C++, Version 4.1)
- TEST32U.EXE (Unicode sample for Microsoft Visual C++, Version 4.1)

TEST can be compiled with Microsoft C++. The 32-bit version of the sample program TEST32 was generated by Microsoft’s AppWizard utility program. This application demonstrates the basics of using the Microsoft Foundation classes and also is a starting point for writing your application.

Before you rebuild an executable version of TEST.EXE (16-bit sample for Microsoft Visual C++, Version 1.52), move the file TEST.MAK to REL16\TEST.MAK.

In this manual, refer to “Sample Programs” for the location of installed samples, and to “Setup Information” for general information on setting up servers and clients.

This following table lists the files provided for the TEST sample.

### Component Files of the TEST Sample

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEST32.MAK</td>
<td>A project file that is compatible with the Visual C++ Workbench and also is compatible with the nmake program provided with the Professional Edition of Visual C++. To build a debug version of the program from the MS-DOS prompt, type: nmake /f TEST32.MAK</td>
</tr>
<tr>
<td>TEST.H</td>
<td>The main include file for the application. It includes project-specific files (such as RESOURCE.H) and declares the CTestApp application class.</td>
</tr>
<tr>
<td>TEST.CPP</td>
<td>The main application source file that contains the application class CTestApp.</td>
</tr>
<tr>
<td>TEST32.RC</td>
<td>A list of all Microsoft Windows resources used by the program, including the icons, bitmaps, and cursors stored in the RES subdirectory. Use App Studio to directly edit this file.</td>
</tr>
<tr>
<td>TEST ICO</td>
<td>An icon file that is used as the application’s icon. This icon is included by the main resource file TEST32.RC.</td>
</tr>
<tr>
<td>TEST.RC2</td>
<td>A file containing resources that are not edited by App Studio. Initially, the file contains a VERSIONINFO resource that you can customize for your application. Place other non-App Studio editable resources in this file.</td>
</tr>
<tr>
<td>TEST.DEF</td>
<td>A file containing information about the application that must be provided to run with Microsoft Windows. It defines parameters such as the name and description of the application and the size of the initial local heap. The numbers in this file are typical for applications developed with the Microsoft Foundation Class Library. To adjust the default stack size, edit the project file.</td>
</tr>
<tr>
<td>File</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MAINFRM.H</td>
<td>Files containing the frame class CMainFrame, which is derived from CFrameWnd and controls all SDI frame features.</td>
</tr>
<tr>
<td>MAINFRM.CPP</td>
<td>AppWizard creates one document type and one view:</td>
</tr>
<tr>
<td>TESTDOC.H</td>
<td>The document. These files contain your CTestDoc class. Edit these files to add your special document data and to implement file saving and loading (via CTestDoc::Serialize).</td>
</tr>
<tr>
<td>TESTDOC.CPP</td>
<td>CTestDoc objects are used to view CTestDoc objects.</td>
</tr>
<tr>
<td>TESTVIEW.H</td>
<td>The view of the document. These files contain your CTestView class.</td>
</tr>
<tr>
<td>TESTVIEW.CPP</td>
<td>CTestView objects are used to view CTestDoc objects.</td>
</tr>
<tr>
<td>STDAFX.H</td>
<td>Other standard files. These files are used to build a precompiled header (PCH) file named STDAFX.PCH and a precompiled types (PCT) file named STDAFX.OBJ.</td>
</tr>
<tr>
<td>STDAFX.CPP</td>
<td>The standard header file which defines new resource IDs. App Studio reads and updates this file.</td>
</tr>
<tr>
<td>RESOURCE.H</td>
<td>Note: AppWizard uses TODO: to indicate parts of the source code which you should customize or add to.</td>
</tr>
</tbody>
</table>

**To run the sample**

1. Copy CTEST.INI to the base Windows directory (such as C:\WINDOWS or C:\WINNT).
2. Edit the copy so that it reflects a valid setup for a connection to an MSM-Activate server.
3. On the server system, use %RR to load the routine APITEST.RTN which is provided in this directory.
4. To run any version of the sample, either double-click the icon for the file or use a Command window to change to the directory that contains the file and then enter its name. When the program is started, it displays a window that initially is empty.
5. Start the test from the menu-item labeled “Test.”
When the parameters in the file CTEST.INI are incorrect, an error is reported.

When the parameters are correct, the progress of the program is displayed in the window, with the latest output at the top.
Line4 (Java)

This sample is a Java program that demonstrates Java-driven communication with an MSM-Server to play a game at various levels of sophistication.

Background

MSM-Activate provides a comprehensive set of interfaces that allow direct access to routines and data held within MSM-Server systems. MSM-Activate is available as a Windows DLL, a COM object and an ActiveX control, as well as in Java and Java Bean versions. MSM-Activate provides a sophisticated, high-performance interface to a wide range of development tools and languages. In all of these variants, MSM-Activate can be used to access MSM-Server systems via its three main methods:

- Login and Logout, which allow a login into the database based on the information held in the controls’ properties.
- Do method, which runs an MSM routine.
- Xecute, which executes MSM code.

As an example, consider code written using the MSM-Activate/Java for a simple game of four-in-a-line. This is a good example of using Java to produce a graphical interface that can be used with standard MSM code. The MSM-Server code was taken from an example originally written for MSM-Workstation; only the calling method has changed.

To play the four-in-a-line game, a Java applet is downloaded to the browser and serves as the engine to produce the graphical front-end and to accept user interaction. The applet uses MSM-Activate/Java to call routines on the MSM system that calculate where the computer should place its token.

In this manual, refer to “Sample Programs” for the location of installed samples, and to “Setup Information” for general information on setting up servers and clients.

To run the sample

1. Use %RR to load ENGINE.TXT into an MSM-Server system that is running MSM-Activate.
2. Edit the file LINE4.HTML so that the parameters for the applet point to the correct machine IP address and port number. Use a valid username and password.
3. Load the LINE4.HTML file with a web browser that supports Java applications, and then play the game.
Security restrictions

Because of security restrictions which are built into Java, if you download a Java applet from a web site, the applet generally can only open TCP/IP connections back to the machine from which it was downloaded. For example, if you put this demonstration on a web server and view it from a remote machine that does not have the Java version of MSM-Activate installed locally, the machine will download the Java files to run the demonstration from the web server. MSM-Activate for Java can only open a TCP/IP connection back to the machine from which it was downloaded (due to the Java security restriction), so the MSM-Server system must be on the same machine as the web server.

If the Java applet is installed on the machine’s hard disk, then the Netscape browser allows it to open a TCP/IP connection to the local system only, 127.0.0.1. With Microsoft Internet Explorer, a locally-loaded Java class is not permitted to open a TCP/IP connection anywhere unless the class is trusted. A class is trusted if its directory is in the classpath or if the class is from a signed .CAB file. If you are having trouble running a Java sample with Microsoft Internet Explorer, try adding its directory to your classpath. Do this by initializing the environment variable CLASSPATH to the full path to the directory from which you are running the sample. Your Windows system may also need to be rebooted for the new environment variable to take effect.

Users who access this application via a proxy server may find that the application does not work. Such failures are normally due to security options that are configured in your proxy server; you may need to have your network administrator reconfigure the proxy server to allow the use of TCP/IP port number that the demonstration is configured to use. If you accessed this site via an ISP that uses proxy servers, it is unlikely that you will be able to access this Java application.

Refer to “Security” in this chapter for additional information.

Implementation

The Java implementation of MSM-Activate works somewhat differently from other MSM-Activate versions due to constraints of the Java language.

Implementation is contained within a small group of classes. The most important class is called MSMAPI and is distributed as MSMAPI.CLASS. This file contains the core functionality; the others are wrapper classes which are used to pass information to and from MSMAPI.

For this demonstration, first declare a reference to an MSMAPI object, in this case, m_MSMAPI:

```java
public class line4 extends Applet {

    private MSMAPI m_MSMAPI;

    A reference now exists to an MSMAPI object called m_MSMAPI. To create an instance of the object and assign it to the variable, use the following code:

    m_MSMAPI = new MSMAPI();

    You can access the newly created instance of the MSMAPI class and its methods via m_MSMAPI. Because the class has the standard methods found on the other MSM-Activate versions, use the following code to log on to an MSM system:
err = m_MSMAPI.msm_login3(m_Server, m_Port, m_UCI, m_VG, m_Password, m_Timeout, m_Partsize, m_Username, version);

The login method name is msm_login3. The standard parameters, server address, port number, volume group, UCI, and so on, are passed. The method returns a code to specify success or failure.

There are some standard constant values within the class (including MSM_SUCCESS to indicate a successful login):

if (err != MSMAPI.MSM_SUCCESS)
---{
------failComms(err);
------return;
---}

You now have an open connection to an MSM system. In passing parameters to MSM, the access methods differ slightly from other access methods, such as an MSM-Activate/COM connection via Visual Basic. Because of the way Java works, you must put the parameters into a container and pass them via this object. You must first define both the container and the parameters it should contain.

Two other classes, STRREF and DO2PARAMS, are used to communicate between the Java program and MSM. STRREF is the container for passing variables. DO2PARAMS is used to define the structure of the variables and the variables themselves.

Define these two classes to objects:

StrRef version = new StrRef();
DO2PARMS p = new DO2PARMS();

Now build the definition of the structure of the parameters in the object you defined as p. Using the nParams property of the object, specify the number of parameters to be passed:

// parcel request as four parameters
-----p.nParams = 4;
-----p.flag = new boolean[4];
-----p.params = new CharRef[4];

For each parameter, specify whether it is passed by reference or by value:

p.flag[0] = false; // pass by value
p.flag[1] = false; // pass by value
p.params[0] = new CharRef();
p.params[1] = new CharRef();
p.params[2] = new CharRef();
p.params[3] = new CharRef();

After the parameters are set up, place the information that is to be passed into the parameters:

p.params[0].value = m_Board.getBoardArray();
p.params[1].value = String.valueOf(m_Top.getSkill()).toCharArray();
p.params[2].value = new char[1];
p.params[3].value = new char[60];

The object you have constructed holds the variables that are to be passed to MSM. The last step is to have MSM-Activate pass this information through and run or execute MSM code. For this example, use the do2 method to pass the routine entry point and the parameter object:
// run the routine
err = m_MSMAPI.msm_do2("CALC^FOUR", p, ret);

Look at the object to determine what information was passed back from MSM.

Use the logout method to close the connection with MSM:

// disconnect
err = m_MSMAPI.msm_logout();

For additional information, review the Java source code (line4.java) and the MSM-Server source code (engine.txt).
Sample (Java)

This sample Java program illustrates simple connectivity between a Java application in a web browser and an MSM-Server.

**Background**

When you start this demonstration, a window is displayed in which you can enter the parameters needed to establish a connection.

**Note** The Java program must be started from a “URL” that invokes the protocol “http.” When you start the program by double-clicking the file icon of the file sample.html, the protocol “file” is used by default. As a security feature, embedded scripts can only be executed from files that are started using the protocol “http” and that reside in a directory that is marked as “execute enabled.”

In this manual, refer to “Sample Programs” for the location of installed samples, and to “Setup Information” for general information on setting up servers and clients.

**To run the sample**

The following window is displayed.
Enter the relevant parameters in the fields in the right column.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>128.200.1.372</td>
</tr>
<tr>
<td>Port</td>
<td>1667</td>
</tr>
<tr>
<td>UCI</td>
<td></td>
</tr>
<tr>
<td>MGR</td>
<td></td>
</tr>
<tr>
<td>Volgroup</td>
<td></td>
</tr>
<tr>
<td>EDM</td>
<td></td>
</tr>
<tr>
<td>Username</td>
<td></td>
</tr>
<tr>
<td>EDM</td>
<td></td>
</tr>
<tr>
<td>Password</td>
<td></td>
</tr>
<tr>
<td>PEANUTBUTTER</td>
<td></td>
</tr>
<tr>
<td>Auto-Logout after</td>
<td>120</td>
</tr>
<tr>
<td>Seconds Remaining</td>
<td></td>
</tr>
<tr>
<td>Repetition Count</td>
<td>10</td>
</tr>
</tbody>
</table>

Use the buttons on the left side of the window to interact with the MSM-Server.

<table>
<thead>
<tr>
<th>Button</th>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>Result</td>
<td>Successful completion</td>
</tr>
<tr>
<td>Login</td>
<td>Execute String</td>
<td></td>
</tr>
<tr>
<td>Execute</td>
<td>Invoke Function</td>
<td></td>
</tr>
<tr>
<td>Multiple Execute</td>
<td>Parameters for Function</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Output</td>
<td>API Versions:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MSM Server 4.4.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MSM for Windows NT, Version 4.4.0 Beta</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Java Client 4.4.0 Beta</td>
</tr>
<tr>
<td>Multiple Function</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keeplive</td>
<td>Logout</td>
<td></td>
</tr>
</tbody>
</table>
The following window illustrates how to log in.

<table>
<thead>
<tr>
<th>Version</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login</td>
<td></td>
</tr>
<tr>
<td>Xecute</td>
<td></td>
</tr>
<tr>
<td>Multiple Xecute</td>
<td>Invoke Function</td>
</tr>
<tr>
<td>Function</td>
<td>Parameters for Function</td>
</tr>
<tr>
<td>Multiple Function</td>
<td></td>
</tr>
<tr>
<td>Keepalive</td>
<td></td>
</tr>
<tr>
<td>Logout</td>
<td></td>
</tr>
</tbody>
</table>

The following window illustrates execution of a M(UMPS) command.

<table>
<thead>
<tr>
<th>Version</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login</td>
<td></td>
</tr>
<tr>
<td>Xecute</td>
<td></td>
</tr>
<tr>
<td>Multiple Xecute</td>
<td>Xecute String</td>
</tr>
<tr>
<td>Function</td>
<td>Parameters for Function</td>
</tr>
<tr>
<td>Multiple Function</td>
<td></td>
</tr>
<tr>
<td>Keepalive</td>
<td></td>
</tr>
<tr>
<td>Logout</td>
<td></td>
</tr>
</tbody>
</table>

API Versions:
- MSM Server 4.4.0
- MSM for Windows NT, Version 4.4.0 Beta
- Java Client 4.4.0 RC-1

Input:
- 123456

Output:
- 56088
Errors that occur in M(UMPS) code will be reported.

<table>
<thead>
<tr>
<th>Version</th>
<th>Login</th>
<th>Execute</th>
<th>Multiple Execute</th>
<th>Function</th>
<th>Multiple Function</th>
<th>Keepalive</th>
<th>Logout</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Result</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execute String</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w 1/0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Invoke Function</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parameters for Function</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;DIV&gt;&quot;EXECUTE+%APISERV&lt;85&gt;&quot;%API</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Test (Java)

This sample Java program illustrates simple connectivity between a Java class and an MSM-Server using MSM-Activate/Java.

Use the Java Class Viewer to start this program.

In this manual, refer to “Sample Programs” for the location of installed samples, and to “Setup Information” for general information on setting up servers and clients.

To run the sample

1. Prepare the MSM environment by using the utility program ^%RR to load the APITEST.RTN on the server system. The file APITEST.RTN is stored in the subdirectory Samples\CDemo of the MSM-Activate directory.

2. If all defaults can be used, enter the following command to start the demonstration:

   > jview Test.class

   If you do not want to use the server defaults, you must edit and recompile the TEST sample. To change just the IP address and port of the server, use the following syntax:

   > jview.exe test.class <IP ADDRESS> <PORT>
OLEsimp3 and OLEsimp4 (Visual Basic)

This demonstration illustrates how to manipulate local variables in MSM from Visual Basic.

Background

These sample Visual Basic programs illustrate how local variables in MSM may be manipulated from a program created with Visual Basic, using MSM-Activate/COM object access to an MSM-Server or to an Automation Server created with MSM-Workstation.

This program has two versions: OLEsimp3 is the 16-bit version, and OLEsimp4 is the 32-bit version. Although these samples were saved under Visual Basic version 4, the 32-bit sample also will run successfully under Visual Basic version 5. When loading this example under VB5, allow Visual Basic to update the file format.

Although this sample program is similar to the program Simple, the parameters required for making a connection to an MSM-Server are taken from a file (TESTOLE.MCO for a connection to a Workstation-built EXE and TESTTCP.MCO for a connection to an MSM-Server).

In this manual, refer to “Sample Programs” for the location of installed samples, and to “Setup Information” for general information on setting up servers and clients.

To run the sample

1. Make copies of the .MCO files.
2. Edit the copies to reflect the parameters for the current environment.

For a connection to an MSM-Workstation Automation server, the name of the executable file must be the full path name for a file containing an application that was created as an Automation Server with MSM-Workstation, by setting the appropriate Make EXE option:

For a connection to an MSM-Server, the IP address, port number, user name and password are the most important parameters.
3. To run this program, start the appropriate copy (OLEsimp3 for a 16-bit environment and OLEsimp4 for a 32-bit environment).

4. In the upper-left corner of the window, select Link and then select Connect.

5. This option opens a pop-up file selection window. Select the appropriate file for the connection.

An appropriate file for a connection to an Automation Server EXE could be:

An appropriate file for a connection to an MSM-Server could be:

6. Once the connection is made, the buttons become active. Press the button labeled “Get” to fetch the value of the selected variable and display its value. For example:

(When the value contains more characters than can fit in the display box, place the cursor in the display box, and use the left and right arrows to scroll horizontally through the value.)
Detail (Visual Basic)

This sample Visual Basic program illustrates how to use MSM-Activate/API to retrieve information from an MSM database.

Background

The Detail application uses the `msm_xecute` function to add and retrieve personal details from an MSM database. The application uses the following form:

```
<table>
<thead>
<tr>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forename: John</td>
</tr>
<tr>
<td>Address: 1 High St</td>
</tr>
<tr>
<td>Phone: 01734 123456</td>
</tr>
</tbody>
</table>
```

Details are stored in the database using the surname key value in the following format:

```
^DETAILS(<SURNAME>)=<FORENAME>*<AGE>*<ADDR1>*<ADDR2>*<ADDR3>*<PHONE>
```

When a new entry is added, the application concatenates the non-key values with an * (asterisk) delimiter character and builds the following M string to execute:

```
S ^DETAILS("SMITH")="John*26*1 High St*Reading*Berks*01734 123456"
```

To retrieve information from the database, the application uses the M WRITE command to return all of the details based on using the surname key:

```
W ^DETAILS("SMITH")
```

The returned information then is extracted from the string using a user-defined Visual Basic PIECE function and is inserted into the correct form fields.

In this manual, refer to “Sample Programs” for the location of installed samples, and to “Setup Information” for general information on setting up servers and clients.
To run the application

To connect to an MSM system
1. Use the Link/Connect menu option to load the connection form.

![Connection Form](image)

2. Enter the details of your MSM-Server system, and then select OK.
   If the connection is successful, a dedicated M server is started, and the system can be manipulated. If the connection is unsuccessful, an error message is displayed in the status text control.

Loading/Running
Use the File/Open Project menu in Visual Basic to open the DETAIL.MAK project that is provided in the Samples\VBDemos subdirectory of the MSM-Activate installation directory. To run the application, select the Run/Start menu option from the Visual Basic design window.

Terminating
Before quitting the application, use the Link/Release menu to make a call to `msm_logout`, thereby closing the connection and terminating the dedicated M server.
Sample32 (Visual Basic)

This sample Visual Basic program illustrates simple connectivity between a 32-bit Visual Basic application and an MSM-Server using MSM-Activate/API.

Background

Since the operational parameters for this demonstration program are hard-coded in the Visual Basic code, the provided executable file (SAMPLE32.EXE) may or may not work in a specific situation.

In this manual, refer to “Sample Programs” for the location of installed samples, and to “Setup Information” for general information on setting up servers and clients.

To run the application

In order to run this demonstration, start Visual Basic, and open the (existing) project. The project description is contained in the file SAMPLE32.VBP.

1. In the project browser window, expand the view on Forms and double-click Form1.

   ![Project Browser](image)

2. In the form-edit window, right-click Login. From the pop-up menu, select View Code. This window displays the parameters that the application uses to connect to the MSM-Server. Change the parameters to reflect the current situation and enter a valid username and password. The “loop-back” IP address cannot be used when a proxy server is being used. In such situations, the actual IP address of the MSM-Server must be specified.

3. When the parameters are correct, click Start on the Visual Basic menu bar. Visual Basic now runs a test image of the window.

4. Click Login to connect to the MSM-Server. When this action is successful, a number lower than 10000 appears in the result box. A number higher than 10000 indicates an error status; refer to “Return Codes” in this manual.
Any M code can now be executed (if the security on the server side has been set up to allow this). The initial sample suggests writing the version number of the MSM-Server.
Any other M code also can be executed.

The return code is equal to 0, which indicates that the request completed successfully.
Simple (Visual Basic)

This sample Visual Basic program illustrates how to use MSM-Activate/API to manipulate local variables in MSM from a program created with Visual Basic.

Background

The Simple application is used to manipulate local variables. The main form, SIMPLE.FRM, has three button controls, three text controls, and three label controls.

- The SetVar control uses `msm_local_set` to set a local variable with a value. The local variable name is obtained from the VarName text control, and the data value is held in the VarValue control.
- The Getvar control uses `msm_local_get` to retrieve data from a local variable. The local variable name is obtained from the VarName text control, and the returned data value is displayed in the VarValue control.
- The Varclr control clears the VarName and RetCode text controls. The RetCode control displays the return code from each function call. A return code of zero indicates success; a non-zero code indicates an error. Refer to “Return Codes” in this manual for a full error code list.

The following is the main form:

In this manual, refer to “Sample Programs” for the location of installed samples, and to “Setup Information” for general information on setting up servers and clients.

To run the application

**To connect to an MSM-Server**

The Simple application uses a separate form named `Linkfrm` to connect to an MSM-Server. Perform the following steps to load this form.

1. From the Simpfrm menu, select Link/Connect.
2. In the text controls, enter the MSM-Server details and then select OK to establish a connection. If the connection is not successful, the system displays an error code in the Status text control.
If the connection is successful, the Linkfrm is unloaded, and the application can manipulate local variables.

**Loading/Running**

Use the File/Open Project menu in Visual Basic to open the SIMPLE.MAK project that is included in the Samples\VB Demos subdirectory of the MSM-Activate installation directory. To run the application, select the Run/Start menu option from the Visual Basic design window.

**Terminating**

Before you quit the Simple application, use the Link/Release menu to make a call to `msm_logout`, thereby closing the connection and terminating the dedicated M server.
OLEDemo (Delphi)

This sample Delphi program illustrates simple connectivity between a Delphi application and an MSM-Server or an MSM-Workstation-built Automation server using MSM-Activate/COM. This sample program has been tested on Delphi Version 3.

Background

The example project oledemo displays a simple form in which you can type a string that is executed on the MSM system. The return value is displayed in another box.

In this manual, refer to “Sample Programs” for the location of installed samples, and to “Setup Information” for general information on setting up servers and clients.

To run the sample

To run this sample program from Delphi, first load the OLEDEM.DPR project file. When this project file is loaded, an edit window containing the file OLEDEMO.PAS will be opened.

In this file, edit the initialization part:

```pascal
MConnector:=CreateOleObject('M.Command');
// This is where you put the name of your MCO file
MConnector.LoadFromMCO('c:\msmole\delphi\local.mco');
MConnector.Login('Password');
```

Replace the name of the file c:\msmole\delphi\local.mco so that the command LoadFromMCO refers to a valid MCO file which points to an MSM-Activate Server or an MSM-Workstation-built Automation server. (Refer to the sample programs OLEsimp3 and OLEsimp4 for more information on MCO files.)

To execute the program, press F9. When the form appears, you can enter an M command in the text box at the top of the form. Pressing the button labeled “Xecute” causes the command to be executed on the server, and the result to be displayed in the edit box labeled “Result.”
Appendices

Return Codes

The following table is a complete list of the MSM-Activate/API return codes, which also are defined in the MSMAPI.H header file. In the first column, the corresponding code number is provided in parentheses under each return code. The second column contains the hexadecimal value for each return code.

<table>
<thead>
<tr>
<th>Return Code and Code Number</th>
<th>10000 +Hex</th>
<th>Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSM_SUCCESS (0)</td>
<td>N/A</td>
<td>Successful completion</td>
<td>The function returned successfully.</td>
</tr>
<tr>
<td>MSM_BASE_ERROR (10000)</td>
<td>#0</td>
<td>Base error code</td>
<td>All errors start from this base error.</td>
</tr>
<tr>
<td>MSM_SOCKET_LOGIN (10001)</td>
<td>#1</td>
<td>Socket error on login server</td>
<td>Unable to connect to the login server. Verify that the server is running and the address is correct.</td>
</tr>
<tr>
<td>MSM_SOCKET_SERVER (10002)</td>
<td>#2</td>
<td>Socket error on M server</td>
<td>Unable to connect to the dedicated server. Verify that sufficient MSM partitions are available to start a dedicated server.</td>
</tr>
<tr>
<td>MSM_NETINIT (10003)</td>
<td>#3</td>
<td>msm_initialize() not called</td>
<td>Call msm_initialize() before using msm_login() for the first time.</td>
</tr>
<tr>
<td>MSM_NETDOWN (10004)</td>
<td>#4</td>
<td>Underlying net system failed</td>
<td>The underlying network subsystem failed.</td>
</tr>
<tr>
<td>MSM_SDNUM (10005)</td>
<td>#5</td>
<td>No more socket descriptors</td>
<td>No additional socket descriptors are available in the WinSock implementation. Close one of the connections and then retry.</td>
</tr>
<tr>
<td>MSM_CONNERR (10006)</td>
<td>#6</td>
<td>Connection error</td>
<td>Non-specific problem with connection.</td>
</tr>
<tr>
<td>Return Code and Code Number</td>
<td>10000 +Hex</td>
<td>Description</td>
<td>Explanation</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>MSM_SDERR (10007)</td>
<td>#7</td>
<td>Invalid descriptor</td>
<td>An invalid system descriptor was passed. Either a connection was not established with this descriptor, or the connection was terminated with msm_logout().</td>
</tr>
<tr>
<td>MSM_INVPARTSIZE (10008)</td>
<td>#8</td>
<td>Invalid partition size</td>
<td>The partition size must be greater than 20 and less than MSMK_MAXPSIZE. This code also is returned from msm_partsize() if the specified partition size cannot be obtained.</td>
</tr>
<tr>
<td>MSM_PASSWD (10009)</td>
<td>#9</td>
<td>Invalid MSM-Activate/API login password</td>
<td>The supplied password does not match the server system's password.</td>
</tr>
<tr>
<td>MSM_TCPLIB (10010)</td>
<td>#A</td>
<td>WinSock library not found</td>
<td>The WinSock library was not found in the Windows directory.</td>
</tr>
<tr>
<td>MSM_TCPVER (10011)</td>
<td>#B</td>
<td>Invalid WinSock version</td>
<td>The WinSock library must be Version 1.1 or later.</td>
</tr>
<tr>
<td>MSM_LOCAL_CONN (10012)</td>
<td>#C</td>
<td>Local communications error</td>
<td>An error occurred while connecting to the local server. Either a connection is already established, or the local server was not started in MSM.</td>
</tr>
<tr>
<td>MSM_BLOCKING (10013)</td>
<td>#D</td>
<td>WinSock blocking</td>
<td>A previous MSM-Activate/API call has not yet returned.</td>
</tr>
<tr>
<td>MSM_NOPART (10014)</td>
<td>#E</td>
<td>No server partition</td>
<td>No partition was available to start the dedicated server.</td>
</tr>
<tr>
<td>MSM_LICENSE (10015)</td>
<td>#F</td>
<td>License expired</td>
<td>The DLL license expired.</td>
</tr>
<tr>
<td>MSM_TIMEOUT (10016)</td>
<td>#10</td>
<td>Read timeout</td>
<td>A read timeout occurred before a response was received from the MSM-Server.</td>
</tr>
<tr>
<td>MSM_INCOMPLETEDATA (10017)</td>
<td>#11</td>
<td>Incomplete data returned</td>
<td>The data returned is incomplete.</td>
</tr>
<tr>
<td>MSM_VXD_ERROR (10018)</td>
<td>#12</td>
<td>Local communications error</td>
<td>An error occurred with the local communications system.</td>
</tr>
<tr>
<td>MSM_APIWIN (10019)</td>
<td>#13</td>
<td>APIWIN not started</td>
<td>The APIWIN.EXE must be started before local MSM-Activate/API calls.</td>
</tr>
<tr>
<td>MSM_GDBSIZE (10020)</td>
<td>#14</td>
<td>Invalid GDB size</td>
<td>The size parameter is incorrect.</td>
</tr>
<tr>
<td>MSM_GDB (10021)</td>
<td>#15</td>
<td>Invalid GDB</td>
<td>The global data block is invalid.</td>
</tr>
<tr>
<td>MSM_LOCKMODE (10022)</td>
<td>#16</td>
<td>Invalid lock mode</td>
<td>The lock mode parameter is invalid. The parameter must be 0, 1, or 2.</td>
</tr>
<tr>
<td>Return Code and Code Number</td>
<td>10000 +Hex</td>
<td>Description</td>
<td>Explanation</td>
</tr>
<tr>
<td>----------------------------</td>
<td>------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>MSM_NOSIBLING (10023)</td>
<td>#17</td>
<td>No other siblings</td>
<td>No global nodes are available when using msm_global_order or msm_global_query.</td>
</tr>
<tr>
<td>MSM_SDB (10024)</td>
<td>#18</td>
<td>Invalid SDB</td>
<td>The subscript data block is invalid.</td>
</tr>
<tr>
<td>MSM_SUBPOS (10025)</td>
<td>#19</td>
<td>Invalid subscript position</td>
<td>The position parameter is invalid.</td>
</tr>
<tr>
<td>MSM_PARTSIZE (10026)</td>
<td>#1A</td>
<td>Partition size not given</td>
<td>The requested position size was not granted.</td>
</tr>
<tr>
<td>MSM_LTIMEOUT (10027)</td>
<td>#1B</td>
<td>Lock timeout</td>
<td>The msm_global_lock command timed out without locking the requested global.</td>
</tr>
<tr>
<td>MSM_INVSUBSCR (10028)</td>
<td>#1C</td>
<td>Invalid subscript</td>
<td>The subscript contains invalid data.</td>
</tr>
<tr>
<td>MSM_NAME (10029)</td>
<td>#1D</td>
<td>GDB and SDB zero length</td>
<td>The GDB and SDB structures were both passed with zero length fields.</td>
</tr>
<tr>
<td>MSM_NOSUCHUCI (10030)</td>
<td>#1E</td>
<td>Invalid UCI</td>
<td>The specified UCI does not exist. The UCI must be three uppercase characters or a NULL pointer.</td>
</tr>
<tr>
<td>MSM_NOSUCHVOLGRP (10031)</td>
<td>#1F</td>
<td>Invalid volume group</td>
<td>The specified volume group does not exist. The volume group must be three uppercase parameters or a NULL pointer.</td>
</tr>
<tr>
<td>MSM_NULLSUBSCR (10032)</td>
<td>#20</td>
<td>Empty subscript</td>
<td>A subscript cannot be passed as an empty string.</td>
</tr>
<tr>
<td>MSM_ARGCNT (10033)</td>
<td>#21</td>
<td>Incorrect number of arguments</td>
<td>The number of arguments passed is invalid.</td>
</tr>
<tr>
<td>MSM_STRLEN (10034)</td>
<td>#22</td>
<td>String too long or empty</td>
<td>The string passed is either greater than MSMK_STRING_LENGTH or is passed as a NULL pointer.</td>
</tr>
<tr>
<td>MSM_UNSPECIFIED (10035)</td>
<td>#23</td>
<td>Unspecified error</td>
<td>An unknown error occurred in the MSM system.</td>
</tr>
<tr>
<td>MSM_NOPORT (10036)</td>
<td>#24</td>
<td>No dedicated port available</td>
<td>All dedicated ports on the server appear to be in use.</td>
</tr>
<tr>
<td>MSM_RETURNLEN (10037)</td>
<td>#25</td>
<td>Returned data too long</td>
<td>The returned data exceeded the buffer size and was truncated.</td>
</tr>
<tr>
<td>MSM_SECURITY (10038)</td>
<td>#26</td>
<td>Invalid response from server</td>
<td>There was a client/server security failure.</td>
</tr>
<tr>
<td>MSM_PROTOCOL (10039)</td>
<td>#27</td>
<td>Server response does not conform to protocol</td>
<td>The format of a message from the server was invalid.</td>
</tr>
<tr>
<td>MSM_DO2PARAMS (10040)</td>
<td>#28</td>
<td>DO2PARAMS structure incorrectly filled-in</td>
<td>There is an inconsistency within the DO2PARAMS structure. Refer to the documentation and sample code.</td>
</tr>
<tr>
<td>Return Code and Code Number</td>
<td>10000 +Hex</td>
<td>Description</td>
<td>Explanation</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------</td>
<td>------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MSM_CONNECTED (10041)</td>
<td>#29</td>
<td>Already connected to server</td>
<td>MSM-Activate/Java only. Indicates an attempt to login when object is already connected to a server, or an attempt to logout when object is not connected.</td>
</tr>
<tr>
<td>MSM_SOCKET_ERROR (10042)</td>
<td>#2A</td>
<td>Unspecified error using a socket</td>
<td>MSM-Activate/Java only. An unspecified error occurred during a socket operation.</td>
</tr>
<tr>
<td>MSM_ARRAY (10043)</td>
<td>#2B</td>
<td>Could not send array, must be an array of variants</td>
<td>Verify that array is at least one dimension in size and consists of types that can be sent.</td>
</tr>
<tr>
<td>MSM_PSIZE (10044)</td>
<td>#2C</td>
<td>Partition size</td>
<td>Attempt to login with an invalid partition size.</td>
</tr>
<tr>
<td>MSM_VERSN (10045)</td>
<td>#2D</td>
<td>Invalid version</td>
<td>The server cannot connect to this version of the client. Either the client or the server will need to be updated.</td>
</tr>
<tr>
<td>MSM_INVRC (10046)</td>
<td>#2E</td>
<td>Invalid record type</td>
<td>This record type is not allowed due to the server's security restrictions on which commands can be used.</td>
</tr>
<tr>
<td>MSM_ALLOW (10047)</td>
<td>#2F</td>
<td>Do command not allowed</td>
<td>The Do entry point is not on the list of allowed entry points. Change security options of server to fix this.</td>
</tr>
<tr>
<td>MSM_UTFCONV (10048)</td>
<td>#30</td>
<td>Cannot convert between Unicode and UTF8 format</td>
<td>Either a Unicode character was supplied that could not be converted to UTF-8, or the UTF-8 sequence from the server is not a valid Unicode character.</td>
</tr>
<tr>
<td>MSM_M_ERROR (10049)</td>
<td>#31</td>
<td>Base M error code</td>
<td>All errors that signify an M error are returned above this base code.</td>
</tr>
<tr>
<td>MSM_SBSCR (10050)</td>
<td>#32</td>
<td>Invalid subscript</td>
<td>The subscript is invalid; either it is empty or contains $C(0).</td>
</tr>
<tr>
<td>MSM_NAKED (10051)</td>
<td>#33</td>
<td>Invalid naked reference</td>
<td>Access to the global variable that uses the naked indicator is invalid.</td>
</tr>
<tr>
<td>MSM_PROT (10052)</td>
<td>#34</td>
<td>Protection error</td>
<td>A user without proper authorization attempted to access a global.</td>
</tr>
<tr>
<td>MSM_UNDEF (10053)</td>
<td>#35</td>
<td>Undefined variable</td>
<td>An attempt was made to reference a nonexistent local or global variable.</td>
</tr>
<tr>
<td>MSM_PGMOV (10054)</td>
<td>#36</td>
<td>Insufficient partition memory</td>
<td>There is insufficient memory space left in the partition to complete the operation.</td>
</tr>
<tr>
<td>MSM_STKOV (10055)</td>
<td>#37</td>
<td>Stack overflow</td>
<td>The system stack overflowed because of nested indirection, program loop, etc.</td>
</tr>
<tr>
<td>Return Code and Code Number</td>
<td>10000 +Hex</td>
<td>Description</td>
<td>Explanation</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------</td>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MSM_SYNTX (10057)</td>
<td>#39</td>
<td>Invalid syntax</td>
<td>The interpreter encountered a syntax error in a line during execution.</td>
</tr>
<tr>
<td>MSM_NOPGM (10058)</td>
<td>#3A</td>
<td>No program</td>
<td>A reference was made to a nonexistent program.</td>
</tr>
<tr>
<td>MSM_MXSTR (10059)</td>
<td>#3B</td>
<td>Maximum string length exceeded</td>
<td>The value of a string exceeds the maximum length allowed by the system.</td>
</tr>
<tr>
<td>MSM_DKSER (10060)</td>
<td>#3C</td>
<td>Invalid database block</td>
<td>A block read from disk was not of the expected type.</td>
</tr>
<tr>
<td>MSM_DKHER (10061)</td>
<td>#3D</td>
<td>Unrecoverable disk error</td>
<td>An unrecoverable error occurred while the system either was reading from or writing to disk.</td>
</tr>
<tr>
<td>MSM_CMMND (10062)</td>
<td>#3E</td>
<td>Illegal command</td>
<td>The interpreter encountered an illegal command.</td>
</tr>
<tr>
<td>MSM_FUNCT (10063)</td>
<td>#3F</td>
<td>Function error</td>
<td>The interpreter encountered an undefined or improperly used function.</td>
</tr>
<tr>
<td>MSM_INDER (10064)</td>
<td>#40</td>
<td>Indirection error</td>
<td>The interpreter encountered an illegal or incorrect use of the indirection operator.</td>
</tr>
<tr>
<td>MSM_DIVER (10066)</td>
<td>#42</td>
<td>Divide by zero</td>
<td>An attempt was made to divide by zero.</td>
</tr>
<tr>
<td>MSM_DKFUL (10067)</td>
<td>#43</td>
<td>Volume group full</td>
<td>No additional space is available on the disk within current UCI limits.</td>
</tr>
<tr>
<td>MSM_LINER (10068)</td>
<td>#44</td>
<td>Line does not exist</td>
<td>A reference was made to a nonexistent line.</td>
</tr>
<tr>
<td>MSM_MERGE (10069)</td>
<td>#45</td>
<td>Merge error</td>
<td>The MERGE destination operand is a descendant of the source operand.</td>
</tr>
<tr>
<td>MSM_MODER (10070)</td>
<td>#46</td>
<td>Device mode error</td>
<td>An attempt to access a device is inconsistent with the open parameters.</td>
</tr>
<tr>
<td>MSM_MXMEM (10071)</td>
<td>#47</td>
<td>VIEW outside limits</td>
<td>A memory address supplied to the VIEW command or $VIEW function is outside the limits allowed by MSM.</td>
</tr>
<tr>
<td>MSM_MXNUM (10072)</td>
<td>#48</td>
<td>Value too large</td>
<td>The value of a number is greater than the largest number allowed.</td>
</tr>
<tr>
<td>MSM_NODEV (10073)</td>
<td>#49</td>
<td>Open invalid device</td>
<td>An attempt was made to open a device that is not defined to the system.</td>
</tr>
<tr>
<td>MSM_NOPEN (10074)</td>
<td>#4A</td>
<td>Use of an unopened device</td>
<td>An attempt was made to use a device that was not previously opened.</td>
</tr>
<tr>
<td>MSM_SYSTM (10075)</td>
<td>#4B</td>
<td>MSM internal error</td>
<td>The MSM system encountered an internal error. Reboot MSM as soon as possible.</td>
</tr>
<tr>
<td>Return Code and Code Number</td>
<td>10000 +Hex</td>
<td>Description</td>
<td>Explanation</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------</td>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MSM_DDPER (10076)</td>
<td>#4C</td>
<td>DDP server error</td>
<td>The DDP server encountered an error while processing a transaction through the network.</td>
</tr>
<tr>
<td>MSM_BADCH (10077)</td>
<td>#4D</td>
<td>Invalid Kanji</td>
<td>The system encountered an invalid Kanji or Shift-JIS character.</td>
</tr>
<tr>
<td>MSM_ASYNC (10078)</td>
<td>#4E</td>
<td>RVG synchronous error</td>
<td>An error occurred during the asynchronous processing of a previous SET or KILL operation on an RVG.</td>
</tr>
<tr>
<td>MSM_BKERR (10079)</td>
<td>#4F</td>
<td>BREAK command</td>
<td>The interpreter encountered a BREAK command.</td>
</tr>
<tr>
<td>MSM_DKRES (10080)</td>
<td>#50</td>
<td>Addressing reserved area</td>
<td>The system attempted to allocate a block in the reserved area of the user's expansion area because the expansion area is almost full.</td>
</tr>
<tr>
<td>MSM_DPARM (10081)</td>
<td>#51</td>
<td>Parameter problem</td>
<td>Either a parameter list was used incorrectly or an invalid use of the QUIT command occurred.</td>
</tr>
<tr>
<td>MSM_EXPER (10082)</td>
<td>#52</td>
<td>Exponential error</td>
<td>An exponentiation error was detected.</td>
</tr>
<tr>
<td>MSM_INHIB (10083)</td>
<td>#53</td>
<td>Network database access inhibited</td>
<td>An attempt to access a database through the network failed because database reads or writes are inhibited on the specified machine.</td>
</tr>
<tr>
<td>MSM_ISYNT (10084)</td>
<td>#54</td>
<td>Illegal line of code inserted</td>
<td>An attempt was made to insert an illegal line of code.</td>
</tr>
<tr>
<td>MSM_LCNSE (10085)</td>
<td>#55</td>
<td>License jobs exceeded</td>
<td>The number of jobs exceeds the license limit.</td>
</tr>
<tr>
<td>MSM_MTERR (10086)</td>
<td>#56</td>
<td>Magnetic tape error</td>
<td>The last input or output operation to a tape device caused an error condition.</td>
</tr>
<tr>
<td>MSM_NOSYS (10087)</td>
<td>#57</td>
<td>Nonexistent VG</td>
<td>The volume group name specified in an extended global reference does not exist or is not accessible through the network.</td>
</tr>
<tr>
<td>MSM_NOUCI (10088)</td>
<td>#58</td>
<td>Nonexistent UCI</td>
<td>The UCI name specified in an extended global reference does not exist or is not accessible through the network.</td>
</tr>
<tr>
<td>MSM_PCERR (10089)</td>
<td>#59</td>
<td>Post-conditional error</td>
<td>Either a post-conditional argument is not allowed on the command, or the post-conditional argument is invalid.</td>
</tr>
<tr>
<td>Return Code and Code Number</td>
<td>10000 +Hex</td>
<td>Description</td>
<td>Explanation</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------</td>
<td>----------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>MSM_PLDER (10090)</td>
<td>#5A</td>
<td>Program load error</td>
<td>The compiled code for the specified routine is from an older version of MSM and cannot be executed by the current version.</td>
</tr>
<tr>
<td>MSM_VWERR (10091)</td>
<td>#5B</td>
<td>VIEW buffer not opened</td>
<td>The VIEW buffer is not available.</td>
</tr>
<tr>
<td>MSM_ZCALL (10092)</td>
<td>#5C</td>
<td>Invalid ZCALL</td>
<td>Either the function name specified on the $ZCALL does not exist, or the specified parameters are invalid.</td>
</tr>
<tr>
<td>MSM_ZCERR (10093)</td>
<td>#5D</td>
<td>Routine ZCALL error</td>
<td>Indicates that an error occurred in the routine invoked by a ZCALL function.</td>
</tr>
<tr>
<td>MSM_ZSAVE (10094)</td>
<td>#5E</td>
<td>Line too large to ZSAVE</td>
<td>During compilation, one of the lines in a routine is too large to fit in the disk buffer.</td>
</tr>
<tr>
<td>MSM_ZSVGP (10095)</td>
<td>#5F</td>
<td>Invalid p-code type</td>
<td>A ZSAVE command was issued, but the volume group was not enabled for the p-code type.</td>
</tr>
<tr>
<td>MSM_TPROC (10096)</td>
<td>#60</td>
<td>Transaction processing error</td>
<td>A transaction processing error occurred.</td>
</tr>
<tr>
<td>MSM_DSCON (10097)</td>
<td>#61</td>
<td>Terminal disconnected</td>
<td>The terminal connection was disconnected.</td>
</tr>
<tr>
<td>MSM_CLOBR (10098)</td>
<td>#62</td>
<td>Attempt to overlay current routine</td>
<td>An attempt was made to overlay the current routine.</td>
</tr>
</tbody>
</table>
## Constants

The following table lists the constants that are defined in the MSMAPI.H header file.

### MSM-Activate/API Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSMK_GDB_SIZE</td>
<td>sizeof (struct global_data_</td>
<td>Number of characters in a GDB.</td>
</tr>
<tr>
<td></td>
<td>block)</td>
<td></td>
</tr>
<tr>
<td>MSMK_NAME_LENGTH</td>
<td>8</td>
<td>Maximum length of a global or local variable name.</td>
</tr>
<tr>
<td>MSMK_NUMBER_SUBSCRIPT</td>
<td>84</td>
<td>Maximum number of subscripts.</td>
</tr>
<tr>
<td>MSMK_SDB_SIZE</td>
<td>sizeof (struct subscript_data_</td>
<td>Number of characters in an SDB.</td>
</tr>
<tr>
<td></td>
<td>block)</td>
<td></td>
</tr>
<tr>
<td>MSMK_STRING_LENGTH</td>
<td>4294967296</td>
<td>Maximum string length.</td>
</tr>
<tr>
<td>MSMK_SUBSCRIPT_LENGTH</td>
<td>254</td>
<td>Maximum length of a single subscript.</td>
</tr>
<tr>
<td>MSMK_UCI_NAME_LENGTH</td>
<td>3</td>
<td>UCI name length.</td>
</tr>
<tr>
<td>MSMK_VOLGRP_NAME_LENGTH</td>
<td>3</td>
<td>Volume group name length.</td>
</tr>
<tr>
<td>MSMK_MAXRETURNLEN</td>
<td>4096</td>
<td>Maximum length of returned data.</td>
</tr>
<tr>
<td>MSMK_MAXPSIZE</td>
<td>16384</td>
<td>Maximum partition size.</td>
</tr>
<tr>
<td>MSMM_SIBLING</td>
<td>1</td>
<td>Updates an SDB with the result of a msm_global_order().</td>
</tr>
<tr>
<td>MSMM_PREVIOUS</td>
<td>2</td>
<td>Performs a reverse ORDER.</td>
</tr>
<tr>
<td>MSMK_LOCK</td>
<td>0</td>
<td>Standard lock.</td>
</tr>
<tr>
<td>MSMK_LOCK_INCR</td>
<td>1</td>
<td>Incremental lock.</td>
</tr>
<tr>
<td>MSMK_LOCK_ZALL</td>
<td>2</td>
<td>ZALLOCATE lock.</td>
</tr>
<tr>
<td>MSMK_UNLOCK</td>
<td>0</td>
<td>Standard unlock.</td>
</tr>
<tr>
<td>MSMK_UNLOCK_DECR</td>
<td>1</td>
<td>Decremental unlock.</td>
</tr>
<tr>
<td>MSMK_UNLOCK_ZDEALL</td>
<td>2</td>
<td>ZDEALLOCATE unlock.</td>
</tr>
</tbody>
</table>
Index

S
$DATA 22, 46
$ORDER 53
$QUERY 55

% %APISERV 14, 15, 21

A
.APIMGR utility 7, 10, 15
  Using 10

C
C Structures 18
Collating sequence 55
Command
  Do 9
  Low-level 9
  Xecute 9
Connection
  Disable remote 10
  Enable remote 10
  Establish 14
  Management 14
  Multiple servers 16
  Termination 16
  Version 3 8

D
Data functions 20
  Memory allocation 20
  Dynamic link library (DLL) 27

E
Environment
  Rundown 16
Error messages 20

F
Function,
  Declaring 27

G
Global
Data
  Block (GDB) 18, 22, 31
  Structures 31
Variable 20
  Functions 22

I
Integers
  In 16-bit or 32-bit versions 28

L
Local variable 20
  Functions 21
Locking
  Global node 51
M LOCK 61
max_time 51
Memory allocation 20
MSM_BLOCKING 71
MSM_CL_TIMEOUT 51
msm_do() 24, 37
msm_do2() 25, 39
msm_gdb_clear() 41
msm_gdb_extract() 42
msm_gdb_initialize() 43
msm_gdb_insert() 44
msm_global_data() 46
msm_global_get() 47
msm_global_kill() 49
msm_global_lock() 51
msm_global_order() 53
msm_global_query() 55
msm_global_set() 57
msm_global_set_null() 59
msm_global_unlock() 61
MSM_INCOMPLETEDATA 23, 24, 25
msm_initialize() 14, 29, 63
msm_local_get() 20, 64
msm_local_kill() 65
msm_local_set() 66
msm_login() 67
msm_login2() 67
msm_login3() 14, 20, 30, 67
msm_logout() 70
MSM_NOSIBLING 53, 55
msm_options() 71
msm_partsize() 72
MSM_PGMOV 15, 21, 72
msm_rundown() 16, 73
msm_sdb_clear() 74
msm_sdb_count() 75
msm_sdb_extract() 76
msm_sdb_initialize() 77
msm_sdb_insert() 78
msm_sdb_insert_null() 79
msm_timeout() 80
msm_version() 81
msm_xecute() 23, 82
msm_xecute2() 23, 84
MSM-Activate servers
Configure 7
Multiple 10
List current 11
Terminate 11

MSM-Activate/API
Distribution files
PC 2
UNIX 6
MSM-Activate/API server
MSM-Activate/API for UNIX 32
Archive files 32
Compiling MSM-Activate/API 32
Example applications 32
Using 7
MSM-Activate/COM Automation Server 2, 27
MSMAPI.H header file 13, 32, 125, 132
MSMAPI32.DLL 28
MSMAPIU.DLL 15, 28
MSMK_GDB_SIZE 31
MSMK_SDB_SIZE 31, 77
MSMK_STRING_LENGTH 66

P
Partition size 15, 72
Pattern match
For Do commands 9

R
Reference section syntax 35
Rundown
Environment 16

S
Security
Configure 8
Groups 8
Startup
Automatic 11
Subscript data block (SDB) 18, 22, 31, 74, 75, 78, 79
SYSGEN utility 7

T
TCHAR parameter
Using 36
TCP/IP 10, 63, 71
Initialization 14
Interface
Initializing 29
Socket device 23
Timeout period 15
U
Unicode 2, 7, 15, 17, 36
Upgrading 7
Username/password
  Configuring 8
  Protection 15

V
Visual Basic 27
  Calling functions 28
  Global variables 28
  Server connection from 30

W
WinSock 14, 63, 73

X
Xecute command 9