



The MML Programmer's Guide

Third Edition

Gordano Ltd

The MML Programmer's Guide

Copyright © Gordano Ltd, 1995-2015. All rights reserved. Printed in the United Kingdom.

Published by Gordano Ltd,
Unit 1, Yeo Bank Business Park, Kenn Road, Clevedon, North Somerset, BS21 6UW.

Printing History:

Oct 2002 First Edition

May 2003 Second Edition

April 2015 Third Edition

ISBN

GMS, Gordano and Gordano Ltd and their logos are trademarks of Gordano Ltd.

Many of the designations used by manufacturers and sellers to distribute their products are claimed as trademarks. Where those designations appear in this book, and Gordano Ltd was aware of a trademark claim, the designations have been printed in capitals or initial capitals

Written by Brian Dorricott, John Stanners, Dean Fenton, Jason Hall and Dean Packer.

Copyright © Gordano Ltd, 1995-2015

GMS

WARNING: YOU SHOULD CAREFULLY READ THE LICENCE AGREEMENT PROVIDED WITH THIS MANUAL BEFORE USING THIS SOFTWARE PACKAGE. INSTALLING THE SOFTWARE ONTO YOUR COMPUTER INDICATES YOUR ACCEPTANCE OF THESE TERMS AND CONDITIONS. IF YOU DO NOT WISH TO ACCEPT ALL OF THESE TERMS, YOU SHOULD STOP INSTALLING THIS SOFTWARE NOW AND DESTROY ALL COPIES OF THE SOFTWARE AND ALL MANUALS AND OTHER DOCUMENTS SUPPLIED WITH IT.

NTMail is a registered trademark of Gordano Ltd.

The Gordano Logo is a registered trademark of Gordano Ltd.

Juce is a registered trademark of Gordano Ltd.

NT is a registered trademark of Northern Telecom Ltd.

Windows NT is a trademark of Microsoft Corporation in the USA and other countries.

All other trademarks are acknowledged.

Patents

Gordano owns a number of patents on its software as listed below:

Autoport

Gordano's "Autoport" technology is patented in the United Kingdom under patent number GB2391649.

A patent application has been filed in the United States and is pending approval.

Maintaining software and data (Automatic Updates)

Gordano's "Maintaining software and data" technology is patented in the United Kingdom under patent number GB2374163.

A patent application has been filed in the United States and is pending approval.

Anti-spam filter (Sender Verification)

Gordano's "Anti-spam Filter" technology is patented in the United Kingdom under patent number GB2385965 and in the United States under patent number 7574476.

Transitory E-mail Addresses

Gordano's "Transitory E-mail Address" technology is patented in the United Kingdom under patent number GB2398399.

Table of Contents

1	Introduction	1
1.1	Who Should Read this Guide?	1
1.2	Other Gordano Guides	1
1.3	Conventions	3
2	Introducing Mail Meta Language	5
2.1	Language Structure	6
	Mixing HTML and code	6
	Pages	6
	Comments	7
	Escape sequences	7
2.2	Types of Script	7
	GUI scripts	7
	GMS Anti-Spam scripts	7
	Timed events	9
	List messages containing executable MML	9
2.3	Program Limits	11
2.4	Directories and Files	11
2.5	The # Operator	11
2.6	Expression Evaluation	12
2.7	Notes on the Script Server	12
2.8	Account Names	12
2.9	Variables	14
	Variable manipulation	14
	Variable search order	15
	Variable types	15
	Object variables	16
	Persistent variable storage	17
2.10	Processing Image Maps	17
2.11	Session Setup	19
2.12	Tips for C and VisualBasic Programmers	19
3	Commands	21
3.1	Def	21
3.2	Do	22
3.3	End	22
3.4	For	22
3.5	If..Else	23
3.6	Include	24
3.7	Return	24
3.8	Session (also called Global)	24
3.9	While	25
3.10	Connect	25
3.11	Destroy	25
4	Function Groups	27
4.1	Messages	28

4.2	User Functions.....	29
4.3	Membership Database Functions.....	30
4.4	File and Directory Functions.....	31
4.5	Folder (Mailbox) Functions.....	32
4.6	Connection Functions.....	33
4.7	String Functions.....	34
4.8	List String Functions.....	35
4.9	Bit and Bit Mask Functions.....	36
4.10	Times, Date and Event Functions.....	37
5	Functions.....	39
	Introduction.....	39
	Return values.....	39
	Files and absolute paths.....	40
	Directories.....	40
	AddAlias.....	41
	AddDomain.....	42
	AddHTTPHeader.....	44
	AddPostfix.....	45
	AddrInRange.....	46
	AddSession.....	47
	AddTimed.....	48
	AddUser.....	50
	AddZip.....	52
	AllowAbsoluteFileNames.....	53
	ArchiveAddMessage.....	54
	ArchiveRecalculate.....	55
	ArchiveRefresh.....	56
	Asc.....	57
	AutoConnect.....	58
	AutoDisconnect.....	60
	BitAnd.....	61
	BitIsSet.....	62
	BitMaskIsSet.....	63
	BitReset.....	64
	BitSet.....	65
	Bound.....	66
	CheckPassword.....	67
	CheckServer.....	68
	CheckServiceAccess.....	69
	CheckTopLevelScript.....	70
	Chr.....	71
	CloseMemberDB.....	72
	CloseZip.....	73
	CollectFromPOP.....	74
	ConfiguredLanguages.....	77
	ConvertCase.....	78
	ConvertForDisplay.....	79
	ConvertToAccount.....	80
	ConvertToDomain.....	82

ConvertToEmailAddress	83
ConvertToFlatHTML	84
ConvertToHTML	85
ConvertToJava	86
ConvertToRealName	87
ConvertToUser	88
ConvertToTime	89
CreateDirectory	90
CreateSetup	91
Date	92
DateTimeFormat	93
DefaultLanguage	94
DefaultListParm	95
DelAlias	96
DelDir	97
DelDomain	98
DelDomainFiles	99
DeleteMemberRecord	100
DelFile	101
DelSession	102
DelTimed	103
DelUser	104
DelUserFiles	105
DiffDate	106
DirSize	107
EncryptPassword	108
EnumRasEntries	109
ExistDomain	110
ExistFile	111
ExistUser	112
ExistVar	113
FileClose	114
FileCopy	115
FileEOF	116
FileIsBinary	117
Filemd5	118
FileOpen	119
FileReadLine	120
FileReplace	121
FileSize	122
FileVscan	123
FileWriteLine	124
FilterDomainsOfType	125
FilterMsg	126
FilterUsersOfType	127
FindFiles	128
FolderAppendMsg	129
FolderClose	130
FolderDelete	131
FolderExist	132

FolderFlush.	133
FolderGetMessageCount	134
FolderGetNewMessageCount.	135
FolderList	136
FolderModified.	137
FolderMsgCheckStatus.	138
FolderMsgSetStatus	139
FolderMsgUnsetStatus	140
FolderOpen	141
FolderRename	142
GetAllMembers	143
GetConnectionDomain.	146
GetConnectionVariables.	147
GetHostedIps	148
GetHostname.	149
GetHTTPCookie	150
GetHTTPPage	151
GetIPAddress	152
GetLoadsharingServer	153
GetLoadsharingServerList	154
GetLocalDomain.	155
GetLocalIps	156
GetLocalAddr.	157
GetLogonUser	158
GetMailboxName	159
GetMaxThreads	160
GetMemberRecord.	161
GetMXRecord	163
GetOs	164
GetOsStr	165
GetPostFixes.	166
GetProcessorStr	167
GetProtocolText	168
GetProtocolType.	169
GetProxyCacheSize.	170
GetRand.	171
GetRealLogonUser	172
GetRemoteConnectionAddr.	173
GetSessionID	174
GetSessionVariables	175
GetStatus.	176
GetSupportInfo	177
GetUID.	178
GetUsersOfType.	179
IMIsAvailable	181
ImportFolder	182
ImportMembers	183
IMSendMessage.	184
IncrementDate	185
InStr.	186

Interpret	187
IsAbsoluteFilename	188
IsConnected	189
IsDate	190
IsDialupEnabled.	191
IsDomain.	192
IsInteger	193
IsIPAddress	194
IsLoggedIn.	195
IsMemberOfList.	196
IsValidDate	197
IsValidEmailAddress.	198
IsValidPassword.	199
IsValidStr.	200
IsValidUserName	201
IsWildcard	202
KillScript	203
LanguageName.	204
Left.	205
Len	206
ListRunningScripts.	207
ListVersion	208
Location	209
Log	210
LoggedInUsers.	211
LSAppend	212
LSAppend2	213
LSDelete	214
LSDeleteElement.	215
LSElement.	216
LSFind.	217
LSFirstMatch	218
LSLength.	219
LSMatch	220
LSOrder.	221
LSPopElement.	222
LSPushElement	223
LSReplace	224
LSSubset	225
Match.	226
Md5Str	227
MemberFormat.	228
Mid.	230
MsgAddAttachment	231
MsgAddBody	232
MsgAddFile.	233
MsgAddHeader.	234
MsgAddRecipient	235
MsgClose	236
MsgCompose	238

MsgCopy	239
MsgCreate	240
MsgEndOfLines	242
MsgReadFirstLine	243
MsgReadNextLine	244
MsgRemoveHeader	245
MsgSetEncoding	246
MsgSize	247
Nls	248
ODBCInstalled	249
OpenMemberDB	250
OpenZip	252
Print	253
ProxyAgeCache	254
PurgeDNSCache	255
ReadNextMemberRecord	256
RegGetVal	258
RegSetVal	259
RemovePostFix	260
Resolve	261
Right	262
RunExecutable	263
SearchFile	264
SendNotification	265
ServerDSNExists	267
ServerValidUser	268
ServiceStart	269
ServiceStatus	270
ServiceStop	271
SetHTTPCacheable	272
SetHTTPCookie	273
SetHTTPResponseStatus	274
SetLogType	275
SetMemberRecord	276
SetPassword	277
SetScriptPriority	278
SetSessionLanguage	279
Sleep	280
SQLCreateDb	281
SQLExec	282
Time	285
TlsEnabled	286
ToInt	287
Trim	288
UrlDecode	289
UrlEncode	290
VerifyUser	291
WeakDecryptValue	292
WeakEncryptValue	293
WildcardFilterMsg	294

WildCardMatch.	295
WordWrap	296
6 Constants	297
7 Troubleshooting	301
7.1 Diagnostics.	302
7.2 List of Script Errors	302
8 FAQs and Examples	305
8.1 Example Robots	305
8.2 Example Timed Events	306
8.3 Example User Defined GUI	306
8.4 What is an API?	306
8.5 What is MML?	306
8.6 What is a script?	307
8.7 Examples	307
Licence Agreements	313
Installation and Contact Information	323

1 Introduction

This guide describes the GMS Mail Meta Language (MML). It covers the following:

- The fundamentals of MML — the four types of script you can produce, the language structure, its use of variables, etc.
- The commands which make up the language.
- A summary of the main groups of functions which are available for you to use to perform specific tasks like file and user handling.
- A full description of every function, with an example of it in use. (You can also add your own functions.)
- An example MML script.
- A list of the constants used in MML.
- Troubleshooting and error information.

1.1 Who Should Read this Guide?

Anyone who wants to write or modify MML scripts should use this guide. The four types of script you can produce are:

- GUI scripts — modify Gordano's graphical user interface (GUI). For example, you might want to restrict a domain administrator's access to some of the areas they can normally access. You could use MML to block access to these areas.
- GMS Anti-Spam scripts — these are run on messages as they arrive.
- Timed event scripts — initiate an event at a specified time, for example to dial up or to back up your system.
- List messages containing executable MML — these mix HTML and text with embedded MML. This area is still under development.

1.2 Other Gordano Guides

The following guides provide additional information:

- *GMS Administrator's Guide* — describes the master GMS mail messaging solution for Linux, Solaris, AIX, Windows 2000 and Windows NT.
- *GMS Communication Server Guide* — describes the master GMS List server solution for Linux, Solaris, AIX, Windows 2000 and Windows NT.
- *Gordano Reference Guide* — provides detailed technical information for those wishing to use any of the available simple or advanced programmer interfaces. This guide describes all Gordano product Registry parameters, and gives example code for robots and DLLs. It provides full details of the files generated and their formats.

- *GMS User Guide* — provides detailed information enabling ease of use for the users of the system.

1.3 Conventions

The following conventions are used in this guide:

Convention	Used for
Courier	Lines of code.
<i>Italic</i>	Other products, services and guides.
Bold	Function names.
UPPERCASE	Reference to a file, directory, constant or acronym.
<value>	Reference to information you must provide.
[parameter]	An optional parameter of a function.

The following symbols are used in this guide:



Tip — gives optional extra information you may want to act on. You can ignore these if you wish.



Information — gives additional explanation of points. You should read these.



Warning — warns of areas where you could damage some element of your system. You must read these.

2 Introducing Mail Meta Language

This section describes the fundamentals of MML It describes:

- The language structure — how programs begin and end, mixing MML and HTML, use of pages.
- The four different ways to use scripts.
- Limits on variables, functions, etc.
- Special directories which are used in MML.
- The # (hash) operator.
- How expressions are evaluated.
- Notes on the script server.
- How variables are manipulated.
- The form account names can take.
- The order in which variables are processed.
- How image maps are processed.
- Session setup.
- A quick summary of how MML differs from C and Visual Basic.

The next chapters of this manual explain the commands and functions available for use within a program. For a list of the constants used with MML, see "Constants" on page 297.

2.1 Language Structure

A program comprises executable scripts, a series of SESSION declarations, and some function calls. There's a semicolon at the end of every complete command. For scripts which mix MML and HTML, the MML code appears between "<#" and "#>" characters.

A short program might look like the following. This simply outputs an HTML page with the words "Hello world" on it.

```
<# /* This outputs simple text to the browser */
print("Hello world");
#>
```

Mixing HTML and code

HTML and MML code can be freely mixed. Portions of HTML can optionally be included in the output (and can be repeated) by placing them inside the conditional and loop statements. For example:

```
<# if (white == black) { #>
<h1>Today White equals Black!!!</h1>
<#   } else { #>
<h1>As usual, black does not equal white.</h1>
<#   }
#>
Some more HTML code here...
```

This sends the following HTML script to the Web browser for display:

```
<h1>Today White equals Black!!!</h1>
Some more HTML code here...
```



The two variables white and black are actually equal in this case (assuming they are not session or form variables). The interpreter automatically generates new variables when it first finds them mentioned in a script. These are always initialised to empty. So this IF statement is asking if two empty variables are equal to each other — which they are.

Pages

The names of all the pages for the script server end in ".mml" or ".htm". If no page is specified, the "index.mml" or "index.htm" page is loaded. If the specified page does not exist, the page "default.mml" or "default.htm" is loaded instead. Failing this, an error is returned.



In a release of GLMail all these pages are encrypted in a DLL so that users cannot change them.

Comments

C++ style `"/"/` comments are not used. Instead, comments appear between `"/*" and "*/" delimiters, like this:`

```
/* This is a comment */
```

Escape sequences

Within strings you must use “escape” characters, as follows:

- `"\"` to produce `"\"`, a single backslash.
- `\"` to produce `"`, a single quote.

2.2 Types of Script

This section describes the four different ways you can use MML.

GUI scripts

To modify the user interface, you write HTML with embedded MML. The MML code appears between `"<#" and "#>" characters, as described above.`

GMS Anti-Spam scripts

These are pure scripts containing no HTML, so you do not need to use the `"<#" and "#>" characters to start/end your script. These scripts do not output anything to the screen.`

When a message arrives, the script runs against it. An example is filterscripts, which implements the filters you set within the GMS Anti-Spam interface using Content > Global Filters.



When a message is passed through an MML script by GMS Anti-Spam an X-MMLScript header is added to the message.

Other custom scripts can be run at different stages of the message delivery using the Connect > Scripts page in the GMS Anti-Spam interface. The stages are:

- Connect
- HELO\EHLO
- MAIL
- RCPT
- DATA
- End Of Message

For example an end of message script might be used to discover if a message contains a .vbs file and redirect the message if it does. Or a Connect script might be used to reject connections from a certain

IP address. Multiple scripts may be run at each stage of the protocol.

- Three types of special local variables are available for these scripts, Action and rcpt and parameter:

Variable	Meaning
Action	0 = OK. Negative value = warning or redirected. Positive value = failure.
Rcpt	E-mail address to redirect to.
Parameter	Error value.

- The "email" message object. This is a special object that allows you to address the message currently being processed by the script in a number of ways. Any headers in a message can be retrieved using the "email" message object, for instance "email\Subject" and so on. There are also a number of special modifiers. Each of these is described in the table below..

Modifier	Returns
\recipients	list of all recipients
\rcipient	the intended recipient
\replyaddress	intelligently works out where responses go
\msgdate	the date from the message
\date	local date of message
\messageid	as given in the headers
\uidl	unique identifier for message
\status	read, unread, etc.
\size	size of message
\subject	subject of message
\lines	no. of lines in message
\header	all headers of message
\body	complete body of message

Which if any of the above objects are available will very much depend on which stage of the SMTP protocol the script is being run. For example, a connection script is run when a remote host first connects to the SMTP service so absolutely no message objects will be available to this script.

The further on in the protocol the script is being run the greater the number of objects that will become available. The rcpt script will be aware of the message recipients but will not let you change these recipients as the script is called for each recipient in turn. By the time you progress through to an eom

script all of the message objects will be available, and all of these objects can be manipulated.



To return the sender of the message you can use just "mail" with no need to precede it with "email". So in the above example if you wanted to find a match on a message sent by joe@companyA.dom you would use:

if (mail == joe@companyA.dom)

The following example shows how to use "action" and "rcpt" to redirect messages to a different account, in this case the postmaster's:

```
if (email\recipient == "companyA.dom\joe")
{
    action = -1;
    rcpt = "postmaster@companyA.dom";
}
```

This example shows "action" and "parameter" being used to return a message to its sender:

```
if (email\recipient == "companyA.dom\joe'")
{
    action = 2;
    parameter = "500 User does not exist";
}
```

Timed events

These are pure scripts containing no HTML, so you do not need to use the "<#" and "#>" characters to start/end your script. These scripts initiate an event at a specified time but do not output anything to the screen.

Two common uses for these scripts are:

- Controlling dial-up — the script can initiate a call at a set time.
- Backing up — emailing the file setup.txt to another site.

List messages containing executable MML

These scripts mix HTML and text with embedded MML, as in this simple example:

```
Hello <# Print(user); #>
```

```
How are you? <# Print(user); #>
```

These scripts can use two special local variables mail and rcpt.

Variable	Meaning
mail	The address that will be used in the SMTP protocol FROM address.
rcpt	The address that will be used in the SMTP protocol TO address. This is normally the email address that the member has joined the list from.

The rcpt variable is particularly useful and you may want to use this in a footer for your lists to indicated the subscribed address. An example of this might be:

You have subscribed to this list using the email address <# print(rcpt; #>



MML Processing must be enabled for your lists for this to work.

2.3 Program Limits

The following limits are enforced:

Item	Maximum number
Number of functions you can define.	64
Maximum function call depth	32
Function parameters	16
Connection and global variables	Unlimited
Local variables	64
String lengths	Unlimited (if memory)
Integers	32 bit limit
Depth of Include statements	32
Token length	128 bytes

2.4 Directories and Files

MML can access files and directories on your system. For security the interpreter will not allow access outside the standard 'BaseDir' for Gordano Products therefore preventing scripts from reading other system directories and files. There are several "special" directories:

- `///help` - these pages are always available.
- `/domain/user/` - pages that are defined in a user's space.
- `///Script` - pages from the BIN file containing server details.
- `/` - always loads `///index.mml` (as above).

2.5 The # Operator

A variable contained within two "#" signs will be resolved when a script is processed. The "#" is also termed the "hash operator". This example shows its use:

```
User = "joe";
Domain = "GLMail.dom";

if (BitIsSet(#domain#\#user#\Type,AccountTypeForward))
{
    Print("This is a forward account");
};
```

The expression `"#domain#\#user#"` is resolved at runtime into `"GLMail.dom\joe"`.

2.6 Expression Evaluation

The server evaluates expressions using the usual precedence rules. The following operators are provided, listed here in order of highest precedence first:

Operator	Meaning
+, -	Unary positive, unary negative.
*, /, %	Multiply, divide, modulo.
+, -	Add, subtract.
==, >=, <=, <, >, !=	Equality, greater or equal to, less than or equal to, less than, greater than, not equal to.
&&,	Boolean AND, OR.
&	String concatenation operator.

MML supports lazy evaluation of && and || statements in much the same way as in 'C'. So rather than writing code that looks like:

```
if (fileName != " ")
{
    if (ExistFile(fileName))
    {
    }
}
```

You can write

```
if (fileName != " " && ExistFile(fileName))
{
}
```

knowing that the second expression will only get called if the first is TRUE.

The same is true of || statements as well, MML evaluates only far enough to establish if the overall expression will be true.

2.7 Notes on the Script Server

Only POST and SMTP can communicate with the WWW:

- SMTP — this is used for GMS Anti-Spam scripts and user scripts. These are read-only and can be redirected or ignored.
- POST — these are used for LIST scripts. The output from a LIST script is returned to POST to be sent out.

2.8 Account Names

Account names take three forms:

- E-mail addresses — specified as "user@domain.dom" or as "domain.dom!user".
- MML account names — specified as "domain.dom\user".
- Usernames — the system tries to map the IP address a user connects from to a domain. If it succeeds, it then uses that domain name.

In the description of functions, the term "account" can refer to any of these three.

2.9 Variables

This section describes the variables used in MML.

Variable manipulation

The interpreter uses "lazy" variable manipulation. That is, you can be lazy and leave it to try and work out what you mean. Unlike C, in MML you do not need to declare variables or declare what type a variable is. Where possible, the interpreter converts variable types to produce sensible results.

For example, if:

- `s1 = "string".`
- `s2 = "another".`
- `s3 = "s1".`
- `n1 = 3.`
- `d1 = "1997-09-21 14:00:03".`

Then the following results could be expected:

Sum	Result	Description
<code>s1 + s2</code>	0	Adds the values of two strings which are zero.
<code>S1 > s2</code>	TRUE	Compares two strings without taking case into account.
<code>S1 + n1</code>	3	Converts the number into a string for the addition.
<code>D1 + n1</code>	1997-09-24 14:00:03	Advances the date by three days
<code>d1 + s1</code>	"1997-09-21 14:00:03string"	Converts date to standard format string and concatenates the two strings.
<code>s1 & #s3#</code>	"stringstring"	The hash (#) means that the internal expression should be resolved, giving <code>s1 + s1</code> . The same string is then concatenated twice.

Note the following:

- The only characters allowed in variable names are these:
abcdefghijklmnopqrstuvwxyz0123456789_#\
- Variable names are case-insensitive.
- The characters "#" (hash) and "\" are reserved and have special meanings. The hash is described above; see "The # Operator" on page 11. The "\" is used to indicate subsets of variables - also known as "objects" (see below).

Variable search order

The search order for variables is as follows:

1. Constants (the highest priority).
2. Local variables, either within or outside functions. (Those within a function take precedence.) These are used when you do not declare a variable. They are set to 0 or to an empty string.
3. Global and session variables. These control access rights.
4. Connection variables — these are used through URLs or Form variables.

Variable types

There are several types of variable in the interpreter. The differences between these are very important. They can be classified as follows:

Type	Function
Session Variables	<p>A session is the period of time between first accessing the system and the session ending, for example at logoff. The session variables can be declared as they are needed. These variables are declared in the following way:</p> <pre>global MyGlobal;</pre> <p>If at any stage you know a session variable will no longer be required it can be released in the following way:</p> <pre>destroy MyGlobal;</pre>
Form Variables	<p>These are the variables that come directly from the Web browser, plus some additional variables from the local Web browser, for example, the remote IP address of the user.</p> <p>In addition, if any variables are entered on the URL or a form that is POSTed to the Web server, these variables will appear here.</p> <p>The variables which are returned depend on the local Web browser. Those we create are SCRIPT_LENGTH, SCRIPT_NAME, REMOTE_HOSTNAME, LOCAL_HOSTNAME, SERVER_NAME, SERVER_PORT, SERVER_PROTOCOL and SERVER_SOFTWARE.</p> <p>Other values come from the URL and POST action.</p>
Local Variables	<p>The interpreter automatically generates local variables if it finds a variable it does not recognise.</p> <p>These variables are only available to the functions in that routine. Values can be passed to other routines in the function as usual.</p>
Object Variables	<p>This is a set of variables that always contain at least one "\" character. The number of "\" characters determines which group of variables they belong to. These are described below in more detail.</p>

Type	Function
Session Variables	<p>A session is the period of time between first accessing the system and the session ending, for example at logoff. The session variables can be declared as they are needed. These variables are declared in the following way:</p> <pre>global MyGlobal;</pre> <p>If at any stage you know a session variable will no longer be required it can be released in the following way:</p> <pre>destroy MyGlobal;</pre>
Constants	Constants are also available so there's no need to use special values in script. These are listed in "Constants" on page 297.
Special Variables	A list of special variables that gives more information about the current status of the server. See the section on these in "GMS Anti-Spam scripts" on page 7.

Object variables

This table gives more details of the object variables defined above:

Object	Function
Gordano	<p>Gordano objects all have two "\" characters in them. They are all of the form domain-name\user-name\variable-name. Gordano Products have three types of variables internally — global, domain-specific and user-specific. For example, \PostRetryTime is a global variable, net-shop-per.co.uk\FaxCompanyName is a domain variable and net-shop-per.co.uk\brian\Aliases is a user variable.</p> <p>The following special values return a list of parameters:</p> <ul style="list-style-type: none"> • \VariableNames - returns a string of all global variable names. • \DomainNames - returns a string of all domain names. • domain\VariableNames - returns a string of all domain variables. • Domain\UserNames - returns a string of all user names in the domain. • domain\NumberUsers - returns the number of users in that domain. • domain\user\VariableNames - returns a string of all user's variable names. <p><i>The "VariableNames" syntax returns a space separated list of URL Encoded names.</i></p>
SQL	<p>SQL access objects contain a single "\" and use a handle created using the id = SQLQuery(). Once the query has been executed the variables have the following meanings:</p> <ul style="list-style-type: none"> • id\<column-name> - contents of last SQLGet command. • id\ColumnNames - list of all the column names returned by SQLQuery. <p>Once a call is made to SQLClose, all values are destroyed.</p>



Object	Function
Message	<p>A message object is created by the function <code>id = MsgCreate</code>. The object "id" can then be used to obtain the following information:</p> <ul style="list-style-type: none"> • <code>id\<n></code> - line n from the mail message. 1 = first line of body. • <code>id\lines</code> - number of lines in the message. • <code>id\header</code> - returns string of header of the message. • <code>id\body</code> - returns string of the body of the message. • <code>id\size</code> - size of the message bytes. • <code>id\recipients</code> - list of recipients for this mail message. • <code>id\MAIL</code> - the "mail" clause for the message. • <code>id\<header-clause></code> - the header clause requested. <p>Once a call is made to MsgClose, all values are crystallised.</p>
Timed	<p>Sets up a timer event, created by calling "TimedAdd". The events are:</p> <ul style="list-style-type: none"> • <code>Id\datetime</code> - date/time of the first time the event happens. • <code>Id\repeat</code> - how often to repeat the event (in seconds). • <code>Id\script</code> - name of script to run. • <code>Id\next</code> - next time the script is going to be run. • <code>EventNames</code> - a space-separated list of names.
Folder	<p>A folder object is returned by the function FolderOpen and destroyed by FolderClose. A folder is a mailbox. You can access:</p> <ul style="list-style-type: none"> • <code>folderId\NumberOfMessages</code> — number of messages in the mailbox. • <code>folderId\FolderModified</code> — whether or not it has been changed. • <code>folderId\<n></code> — the number of the message in the folder. • <code>folderId\1</code> or <code>folderId\1\mail</code> — you might set "MsgId = folderId\1". Note that this is read-only. Do not close it - use <code>MsgCopy</code> if necessary.

Persistent variable storage

Variables can be stored either at system, domain or user level, it is up to you where you store them. To set them as system variables you just set

`\\variablename=value`

If you want to store on a domain basis use

`domain\\variablename=value`

and on a user basis it is

`domain\user\variablename=value`

Retrieval is similar so for example to retrieve a global variable use

`myvar=\\variablename`

The domains and variablenames are absolute

2.10 Processing Image Maps

The way in which the Web client returns information from image maps makes their use by the Web server difficult in certain

circumstances. For example, if you wish to display lots of "UP" buttons as GIFs (generated by a loop) and then detect which button is pressed by using the URL, you will have problems!

Consider the line:

```
<input type=image src="up.gif" name="map.1">  
<input type=image src="up.gif" name="map.2">
```

When the first image is selected — at (4,6) — the Web client returns two variables:

```
map.1.x = 4  
map.1.y = 6
```

However, we need to know which of the two images was selected, not where in the image the selection was made. Therefore the Web server generates a further variable:

```
map = 1
```

This means we can look at the variable called Map and find out that the first image was selected. This conversion rule is a general rule — any variable with a dot in it has an additional version made with the first part equal to the second.

2.11 Session Setup

A session is established as follows:

1. The Web server waits without any connections.
2. A connection is made to the server, probably from a Web browser.
3. A session is created. At this point nobody has logged on, no user is verified and no global variables have been defined.
4. The session id is allocated in one of three ways:
 - Using the IP address the connection came from.
 - Using a Browser cookie, a string of 100 characters.
 - Using both IP address and Browser cookie. This is the default.
5. When the session is set up, the time is recorded. This is used to check timeouts etc.
6. Session variables are set up as required during the session.
7. The session ends when the user logs off or a timeout occurs.

Note that the logon etc. come *after* the session is created.

If a second connection is made to the server, the server checks to see whether this is already set up by examining its IP address and cookie.

2.12 Tips for C and VisualBasic Programmers

If you are used to C or VisualBasic, note that MML differs from it in several main ways:

- Variable names etc. are case-insensitive.
- Variables are not strictly typed.
- Variable names can be substituted.
- There's no space between "Else" and "If". MML uses "Elseif".
- There are no "++" and "--" operands.
- The first item in a list is item 1, not item 0.

MML, VisualBasic and C all use "\r\n" to produce a CRLF (line feed).

3 Commands

This section describes the following MML commands. These are the building blocks of MML scripts:

- Def — declares a new function.
- Do — executes a statement in a block
- End — terminates processing of the script.
- For — starts a loop.
- If .. Else — if the condition is true executes the first statement, otherwise executes the second statement.
- Include — loads a file.
- Return — returns from a function.
- Session (also called Global) — marks the variables as session variables which will persist until the session is closed.
- While — executes statements in a block while the condition is true.

3.1 Def

Declares a new function. If you cannot find the function you want listed in the following chapter, use this command to define it.

Syntax

```
Def function(x,y,...) { ....statements.... }
```

Remarks

Note that:

- You can use "return" to return a value.
- Do not pass anything but an integer or string to a function. For example, do not pass it a file or file handle.
- You can define up to 64 functions in one session. The maximum call depth (one function calling others) is 32.
- Functions can be used recursively.

Example

```
def PrintMsg(x)
{
    print ("<h1 align=center>", x, "</h1>\r\n");
}

PrintMsg("Hello");
```

Result

```
<h1 align=center>Hello</h1>
```

3.2 Do

Executes all the statements in the statement block until the condition is no longer TRUE. (This means that all the statements will always be executed at least once.)

Syntax

```
do { ....statements.... } while (condition)
```

Example

```
count = 0;
do
{
    print("dinah ");
    count = count + 1;
}
while(count < 7);
print("Batman!");
```

Result

dinah dinah dinah dinah dinah dinah dinah Batman!

See also

While()

3.3 End

Terminates processing of the script and writes all current output.

Syntax

```
end;
```

Example

```
print("Hello<br>");
end;
print("You will never see this!<br>");
```

Result

Hello

3.4 For

On entry to a loop the For command:

1. Executes the "start_statement".
2. Processes the condition and, if true, executes all the statements in the following block.
3. Executes the "end_statement" and re-evaluates the condition.

Syntax

```
for (start_statement; condition; end_statement)
{
    statements...
}
```

Remarks

If the condition is false, execution skips to the end of the loop and continues from there. Note that the open and close statement blocks are mandatory.

Example

```
for (i = 0; i < 10; i = i + 1)
{
    print(i);
}
```

Result

0123456789

3.5 If..Else

If the condition evaluates to a non-zero value, the statement which follows it is executed. The open and close statement block symbols are mandatory.

Syntax

```
if (condition) { statement1; statement2; etc. }

elseif (condition) { statement3, statement4 etc.}

else { statement5; statement6; etc. }
```

Remarks

There's no space between the "Else" and "If" in "Elseif". You can also use "Elif" instead of "Elseif".

Example

This shows use of If and Else with the AddZip function:

```
if (zip)
{
    if (!AddZip(zip, "\\Timed.txt"))
    {
        print(" Couldn't add to zip");
    }
    CloseZip(zip);
}
```

```
    print("Opened zip.");  
  }  
  else  
  {  
    print("Failed to open zip");  
  }
```

3.6 Include

Loads the given file and starts interpreting it.

Syntax

```
include "filename";
```

Remarks

An invalid or non-existent filename generates an error.

Variables are given scope as though the included block of code was copied directly into the place where the Include command appears.

Example

```
include "/index/header.mml";
```

3.7 Return

Returns from a function immediately.

Syntax

```
return;
```

Remarks

Using Return is a poor alternative to encasing a block in an If statement.

Example

```
Def MyFunc()  
{  
  Msg = MsgCreate("dean@test.dom", "fred@copa.dom", "Setup");  
  
  if (!msg)  
    return;  
  
  etc. etc.  
}
```

3.8 Session (also called Global)

Marks the variables as session variables.

Syntax

```
session variable_name, variable_name;
```

Remarks

The session variables persist until the session is closed.

Example

```
session var1, var2;
```

3.9 While

Executes all the statements in the statement block while the condition is TRUE.

Syntax

```
While (condition) { statement }
```

Remarks

If the condition is not true, the statements are not executed at all.

Example

```
while (i < 10)
{
    print(i);
    i = i + 1;
}
```

Result

```
0123456789
```

3.10 Connect

Marks the variables as connection variables.

Syntax

```
connect variable_name, variable_name;
```

Remarks

The connection variables persist until the session is closed or destroyed.

Example

```
connect var1, var2;
```

3.11 Destroy

Destroys variables previously marked as session variables.

Syntax

```
destroy variable_name, variable_name;
```

Remarks

Destroys the session variable.

Example

```
destroy var1, var2;
```


4 Function Groups

This section summarises the main groups of MML functions. It gives an overview of the way you should use related functions.

The groups covered are:

- Messages.
- Users.
- The membership database.
- Files and directories.
- Folders (mailboxes).
- Strings.
- List strings.
- Bits and bit masks.
- Times, dates and events.

Some functions do not fall into any group. These are covered with all the other functions in the following chapter, which describes every function with its syntax, a description and an example of its use.

4.1 Messages

Use the message functions to produce or edit messages. Start by calling **MsgCreate** to return a message handle for use by the other functions.

Use these functions to build a new message:

- **MsgCreate** — starts generation of a new message.
- **MsgClose** — completes creation of a message.
- **MsgCopy** — copies a message. This is used mainly in script.mml, which performs GSM Anti-Spam's message filtering.
- **MsgAddBody** — appends a line to the message body.
- **MsgAddFile** — attaches a file to a message.
- **MsgAddHeader** — adds a new clause to the header.
- **MsgAddRecipient** — adds an RCPT clause.

This example builds a message to send Lou a zip file:

```
Msg = MsgCreate("joe@companyA.dom", "lou@Test.dom", "Your file");
if (Msg)
{
    MsgAddBody(msg, "Hello Lou");
    MsgAddBody(msg, " ");
    MsgAddBody(msg, "Here's the file");
    MsgAddBody(msg, "Joe");
    MsgAddFile(msg, myzip, binary)

    Print(Msg\Recipient, "<br>");
    Print(Msg\Date, "<br>");

    MsgClose(Msg, MSG_SEND);
}
```

Use these functions to edit an existing message:

- **MsgReadFirstLine**, **MsgReadNextLine** — read the first then subsequent lines of a message.
- **MsgEndOfLines** — tests for the end of the message. Use this in a message reading loop.

This example edits a message:

```
line = MsgReadFirstLine(Msg);
while (!MsgEndOfLines(Msg))
{
    Print(ConvertToHTML(line));
    Print("\n");

    line = MsgReadNextLine(Msg);
}
```

4.2 User Functions

Use these functions to produce or edit accounts. You can also encrypt passwords and check that a user's details are valid.

The functions are as follows:

- **AddUser** — adds an account. For the account to start operating, the script must set the TYPE variable for the account using the appropriate constants.
- **AddAlias** — adds an account alias.
- **DelUser** — deletes an account. (Before deleting a domain, you must delete users and their files.)
- **DelUserFiles** — deletes a user's files. Use this before **DelUser**.
- **ExistUser** — checks that a given user exists.
- **FilterUsersOfType** — lists accounts of the specified type.

This example shows :

```
Domain = "GMS.dom";
NewUser = "joe";
ForwardEmail = "joe@companyA.dom";

if (!ExistUser(#domain#\#NewUser#))
{
    if (AddUser(NewUser, Domain, "password"))
    {
        #domain#\#newuser#\Type = AccountTypeMailbox;
        #domain#\#newuser#\Forward = ForwardEmail;
    }
}
```

Other miscellaneous user functions are as follows:

- **GetLogonUser** — returns the session's account name.
- **EncryptPassword** — encrypts an account's password.
- **VerifyUser** — checks that an account and password are valid.

4.3 Membership Database Functions

The membership database records the members of lists. This is a flat file or an SQL database.

The functions use two separate handles:

- The database handle opened using **OpenMemberDb**. This is used by the functions marked below by an asterisk*.
- The member handle opened by **GetMemberRecord** and used by **ReadNextMemberRecord**, **SetMemberRecord** and **MemberFormat**.

Use these functions for membership database management:

- **OpenMemberDB** — opens the specified member database and produces its database handle.
- **CloseMemberDB*** — closes a member database file.
- **GetMemberRecord*** — opens a member record and produces the member handle which the following three calls use.
- **ReadNextMemberRecord*** — read the next record in the file.
- **SetMemberRecord** — sets information on a member.
- **MemberFormat** — extracts variables from a member record.
- **DeleteMemberRecord** deletes a member record from the database.

This example opens a member database and prints the member field of all its records:

```
mydb = OpenMemberDB("test.com\\testlist",FILE_READ);

if (mydb)
{
    while (mymember = ReadNextMemberRecord(mydb))
    {
        Print(" Account: ",mymember\member," <BR>");
    }

    CloseMemberDB(mydb);
}
```

These two do not use the handles so can be used independently:

- **IsMemberOfList** — tests whether a named account is a member of an NTList list.
- **GetAllMembers** — lists the list members. This automatically opens the database, obtains the results then closes the database.

4.4 File and Directory Functions

There are three groups of file functions, as described below. The functions let a verified account use absolute paths.

For reading from/writing to files:

- **FileOpen** — opens a file, returning a file handle for use by the four following functions.
- **FileClose** — closes the file.
- **FileReadLine, FileWriteLine** — read from/write to a file.
- **FileEOF** — tests for the end of the file. Use this in a file reading loop.

This example shows how to use these functions:

```
file = FileOpen(Resolve("#dir#\#filename#"),FILE_READ);
if (file)
{
    while (!FileEof(file))
    {
        Print(FileReadLine(file), "<br>\n");
    }
    FileClose(file);
}
```

For general file management use these:

- **ExistFile** — tests whether a single file exists.
- **FindFiles** — lists files or directories, usually matching a wildcard.
- **SearchFile** — searches a file for the given string.
- **DelFile** — deletes a file.
- **FileReplace** — replaces one file with another.

There are similar functions for use with directories — **CreateDir**, **DelDir** and **DirSize**.

For zip files use these:

- **OpenZip, CloseZip** — open a zip for use by **AddZip**, then close it.
- **AddZip** — adds a file to a ZIP archive.



You cannot use MML to do things like extract files from a Zip file

4.5 Folder (Mailbox) Functions

These functions handle folders (mailboxes). **FolderOpen** returns a folder handle which is used by those functions marked below with asterisks. The other functions use the folder's name directly.

The functions comprise three groups.

Use these for folder handling:

- **FolderOpen** — opens a folder.
- **FolderClose*** — closes a folder.
- **FolderDelete*** — deletes a folder.
- **FolderAppendMsg** — appends a message to a folder.

This example appends a message from Lou's mailbox to Joe's mailbox:

```
folder1 = FolderOpen("companyA.dom\\users\\lou\\inbox.mbx");  
msg = MsgCopy(msg_44);
```

```
folder2 = FolderOpen("companyA.dom\\users\\joe\\inbox.mbx");  
FolderAppendMsg(folder2,msg);
```

```
folderclose(folder1,FALSE);  
folderclose(folder2,TRUE);
```

Use these to obtain folder information:

- **FolderList** — lists folders (you can specify a wildcard).
- **FolderGetMessageCount** — counts the messages in a folder.

Messages have a status setting, one of the "FOLDER_MSG_" constants listed in "Constants" on page 297. The following functions test or control the status of a folder:

- **FolderMsgCheckStatus*** — tests a message's status.
- **FolderMsgSetStatus*** — gives a message the specified status.
- **FolderMsgUnsetStatus*** — removes a status value.

4.6 Connection Functions

These functions control dial-up and connection to POP accounts. They also return information on connections' parameters, IP addresses, etc.

For a dial-up connection use:

- **AutoConnect** starts a dial-up connection.
- **AutoDisconnect** drops the current connection (depending on the timeout period, the connection count and whether the connection was established by GMS).
- **IsConnected** reports the status of a RAS connection.

CollectFromPop collects messages from a POP account for local delivery. You can sort mail based on e-mail address, header clause, etc. and deliver to a single address, a list of addresses, a default address, etc.

For information on connections use:

- **GetConnectionVariables** lists a connection's parameters.
- **GetHostedIps** lists IP addresses of all network cards on the system.
- **GetHostname** performs a reverse lookup on the address specified.
- **GetIPAddress** looks up the given host and returns its IP Address.
- **GetLocalDomain** returns the domain name associated with an IP address.
- **GetLocalIps** returns IP addresses which are local to the network.
- **GetLocalServerAddress** returns the IP addresses which the user's Web browser has connected to the server on.
- **GetRemoteConnAddr** returns the IP address of the Web browser used by the user connecting.

If you use a load sharing array of servers:

- **GetLoadsharingServer** returns the server in the load sharing array which handles the account.
- **GetLoadsharingServerList** lists the servers in the load sharing array.

4.7 String Functions

A string is a set of characters. The string functions let you cut parts of a string out, find substrings within a string, etc.



A string containing a number of separate elements, for example "ab1 ab2 ab3" is termed a list string; see the following section for functions which manipulate the individual elements.

The functions are:

- **Asc** — returns the ASCII value of the first character.
- **Len** — returns the length of the string.
- **Left, Right** — return n characters from the left or right hand side of the string.
- **Mid** — returns n characters from within a string.
- **Match** — compares two strings.
- **Trim** — removes spaces from either end of a string, also tabs, carriage returns and line feeds.

For example, the following function prints "World":

```
z = "World Wide Web";  
Print(left(z, 5));
```

While this prints "14":

```
Print(len("World Wide Web"));
```


4.8 List String Functions

A list string is a string containing a number of separate elements, for example "ab1 ab2 ab3". The elements are normally separated by spaces, but most of the functions can treat strings separated by other characters.

Use the "LS" functions to manipulate list strings as follows:

- **LSAppend** and **LSAppend2** — add elements to a string.
- **LSDelete** and **LSDeleteElement** — cut elements from a string
- **LSElement**, **LSSubset** — obtain one or more elements.
- **LSFind** — report an element's position in the string.
- **LSLength** — counts the number of elements.
- **LSMatch** , **LSFirstMatch** — find elements in a string.
- **LSOrder**, **LSReplace** — change the string in some way.

For example, this function prints "def ghi", obtaining two elements from the list, starting with the second:

```
LSSubset("abc def ghi jkl mno", "def", 2);
```

This function prints "hello def", deleting the second element from the list:

```
LSDelete("hello abc def", 2);
```

4.9 Bit and Bit Mask Functions

The bit handling functions let you compare and change numbers efficiently. All these functions work on integer numbers. The least significant bit is bit 0.

You can use these functions for any bit variables you use. GMS uses bit masks for variables including user type and list member properties; see the *GMS Reference Guide* for details.

The functions are:

- **BitAnd** — performs a bitwise AND on two integer numbers.
- **BitIsSet** — tests whether bit n is set in a number
- **BitMaskIsSet** — tests whether all the mask bits are set in a number.
- **BitReset** — resets bit n in a number to 0.
- **BitSet** — sets bit n to 1.

This example tests whether all the bits in 13, the mask, are also set in 9:

```
if (BitMaskIsSet (9, 13))
{
    Print("1 - All bits are set. ",<br>);
}
```

This example turns off the mailbox bit for an account:

```
#domain#\#user#\Type = BitReset(#domain#\#user#\Type,AccountTypeMailbox);
```

4.10 Times, Date and Event Functions

These functions comprise two groups, as described below.

Use these for event handling:

- **AddTimed** — adds a timed event to the schedule. Use this for dial-up, zipping logs, etc.
- **DelTimed** removes an event from the schedule.

Use these for general date and time use:

- **ConvertToTime** — converts an input string into a standard time format.
- **Date** returns the current date in standard format. Use it to create a date for use elsewhere.
- **DateTimeFormat** takes a date structure and prints it in the format requested
- **Time** returns the current time or outputs the time specified.

For example, this function returns the standard hh:mm:ss time format "19:00:00":

```
ConvertToTime("19:00");
```


5 Functions

This section describes all the script server functions which we provide for you. You can call these from within your MML scripts.

Introduction

In addition to the functions described in this chapter, you can create new functions in two ways:

- Using the script command Def; see "Def" on page 21.
- Using a DLL; see the *GMS Reference Guide* for details.

Note the following:

- Where there are multiple parameters and none of these is marked as optional, you must supply them all, though you can use empty strings.
- Optional parameters appear in [and] brackets.
- Unless stated otherwise, the first element in a list is element 1, not element 0.

Within strings you must use "escape" characters, as follows:

- "\\" to produce "\", a single backslash.
- \" to produce ", a single quote.

Return values

Many functions return a value, indicated in the syntax as shown below for this function which returns a string:

str = Chr(int)

These values are grouped in this chapter as follows:

Value	Meaning
addr	An address.
n	An integer.
str	A string.
t	A time or date/time string.
tf	TRUE/FALSE, used for functions which test something.
user	An account name.
x	Any other variable type.

If a function only returns a TRUE/FALSE value to show whether it succeeded or failed, no return value is shown in the command syntax. For these, check the details for each function under the heading "Return".

Files and absolute paths

As a security precaution, access to absolute paths is restricted. Note the following:

- Anyone can open a file which they can access without specifying its absolute path.
- Accounts which are not verified cannot access absolute paths.
- If the account is verified and the *absolutepath* parameter is set to TRUE, the account can access absolute paths.
- If the account is empty, use the domain's directory. If the domain is empty, use the Gordano base directory.

Directories

Specify a directory as "domain\user\directory name". Both domain and user can be empty.

AddAlias

AddAlias creates an alias for the specified account. Adding aliases is a complex process and requires that both the original and alias accounts be modified and cross-linked correctly. This is particularly complex when you consider that one name may be an alias for two different accounts. Therefore it is strongly recommended that the `AddAlias()` and `DelAlias()` functions are used in order to maintain user database integrity.

AddAlias(account_name,new_alias)



Only a verified account can use this function.

Parameters

Account_name

The account's name in the form "domain\username".

New_alias

The alias to add for this account.

Returns

TRUE if the alias is created successfully, otherwise FALSE.

Remarks

AddAlias works with NT User Database and virtual domains.

Example

This adds the alias "Sales-team" to Joe's aliases:

```
AddAlias("companyA.dom\users\joe","Sales-team");
```

See also

AddUser, DelAlias

AddDomain

AddDomain creates a new GMS domain. Adding a domain to a system is a complex process and the results depend on the domain type being created. The addDomain() function takes the minimum number of parameters required to create a domain. Full and Virtual domains will automatically have a "postmaster" account created. A list of IP addresses available for the full domain may be obtained using the GetHostedIPs() function. On a heavily loaded system, the creation of a domain may take a considerable time (eg. 1 minute). As soon as the function returns, the domain is functioning and can process email and web activity.



*Only a verified account can use this function.
Attempting to create a full domain with an IP address that is already used
will fail*

For a full domain, use:

AddDomain(domain, domain_type, ipaddress)

For a POP or robot domain, use:

AddDomain(domain, domain_type)

For a virtual domain, use:

AddDomain(domain, domain_type, base_domain, postfix)

Parameters

Domain_name

The new domain's name.

Domain_type

DomainTypeFull, DomainTypePOP, DomainTypeRobot or
DomainTypeVirtual.

ipaddress (full domain only)

ip address to be assigned to the domain.

Base_domain (virtual domain only)

The full domain the virtual domain is in. For details, see the
Gordano Administrator's Guide.

Postfix (virtual domain only)

The name used to differentiate users in virtual domains.

Returns

TRUE if the domain is created successfully, otherwise FALSE.

Example

```
if (AddDomain("companyA.dom", DomainTypeFull))
{
    if (AddUser("joe", "companyA.dom", "password"))
    {
        Print("Added user \"joe\" to new domain.<br>");
        Print("Account created on ",companyA.dom\joe\DateCreated);
    }
    else
    {
        Print("Failed to add user to new domain.<br>");
    }
    Print("Domain created on ",companyA.dom\\DateCreated);
}
else
{
    Print("Failed to create domain.<br>");
}
```

Result

Added user \"joe\" to new domain.
Account created on 12-04-99
Domain created on 10-04-99

See also

AddUser, AddSession, GetHostedIPs

AddHTTPResponseHeader

AddHTTPResponseHeader adds the specified text to the HTTP Response header.

AddHTTPResponseHeader(value)

Parameters

Value

the text to add to the HTTP response header.

Returns

Nothing

Example

```
AddHTTPResponseHeader("your text here");
```

See Also

SetHTTPResponseStatus

AddPostfix

AddPostfix adds the specified text as a postfix for the specified domain.

AddPostFix(domain,postfix)

Parameters

domain

the domain the postfix is to be added to.

postfix

the postfix to add to the domain.

Returns

Nothing

Example

```
AddPostFix("CompanyA.dom","postfix1 ");
```

See Also

GetPostFixes, RemovePostFix, AddDomain

AddrInRange

AddrInRange tests whether a specified IP address is within the range specified.

AddrInRange(IPAddressList,IPAddress)

Parameters

IPAddressList

A range of IP addresses, either a space-separated list or a range defined using a wildcard.

IPAddress

The IP address to check.

Remarks

This call supports IP address logic as follows. In all these cases the letters *a* to *e* represent numbers in the range 0 to 255:

- *a.b.c.d* — a specific IP address, for example 194.194.194.194.
- *a.b.c.** — all IP addresses beginning *a.b.c.* For example, 194.194.194.* gives addresses in the range 194.194.194.0 to 194.194.194.255.
- *a.b.c.d-e* — a range of IP addresses from *d* to *e*. For example, 194.194.192-194.* gives addresses in the range 194.194.192.0 to 194.194.194.255
- *a.b.c.d/n* — means use the first *n* bits. For example, 194.194.194.194/22 gives addresses in the range 194.194.192.0 to 194.194.195.255. Similarly, 194.194.194.194/16 gives IP addresses in the B Class range 194.194.0.0 to 194.194.255.255.
- *!a.b.c.d* — the "!" at the beginning of the address means NOT. For example, !194.194.194.194 means not 194.194.194.194.

Returns

TRUE if the address is within the range, otherwise FALSE.

Example

```
if (AddrInRange(194.211.0.*,194.211.0.99))  
{  
    Print("Address is in range.<br>");
```

}Result

Address is in range.

See also

Bound

AddSession

AddSession validates the session. It validates the client, lets them see files outside their base directory and gives them access to all MML functions.

AddSession(account[,password])

Parameters

Account

The account which made the connection.

Password

If the password is supplied and matches correctly, the function gives the session VerifiedScript status.

If no password is supplied, the function adds a session for a non-verified account, for example an anonymous list user. You then use the call **GetLoggedInUser**.

Returns

TRUE if the function succeeds, FALSE if the password fails or the account and domain do not exist.

Remarks

If you access from an unknown connection, we create a session. This means the session already exists when you use this function. In this case, the function simply validates the account.

Example

```
AddSession("companyA.dom\joe", "wh1sky");
```

See also

DelSession, GetLoggedInUser, VerifyUser

AddTimed

AddTimed adds a timed event to the schedule. Use this for dial-up, zipping logs, etc.

AddTimed(name,date,repeat,script,days,start_time,finish_time[,parameters])



Only a verified account can use this function.

Parameters

Name

The name of the event created (a string with no spaces).

Date

The date the first event starts.

Repeat

How often to repeat the event (in seconds).

Script

The script to run. The base directory is the Gordano directory.

Days

A string of digits in the range 0-6, representing days of the week (0 = Sunday). For example "15" = Monday and Friday.

Start_time

The event start time. "01:00" is 1 AM, "23:00" is 11 PM.

Finish_time

The event end time. "01:00" is 1 AM, "23:00" is 11 PM.

Parameters (optional)

A value passed to the script when it is executed. The script accesses it as the variable "parameters".

Returns

TRUE if the event is created successfully, otherwise FALSE.

Remarks

A timed event runs in the context of a logged-on verified account. There's no need to log on.

The schedule list is held in memory by the Web server and written to disk each time it changes as the file `timed.txt`.

This function has the following values that can be accessed using the event name followed by "`\<value>`". Each value can be set and read:

- `EventNames` — list of event names.
- `Id\` — date and time when the event will first happen.
- `Id\repeat` — how often to repeat the event (in seconds).
- `Id\script` — name of script to run.
- `Id\next` — the time the script is next going to be run.

Example

This example creates an event and then prints out all the details on it:

```
if (AddTimed("event", "05-06-99", 60, "\timed.mml", "0123456",
"09:00:00", "17:30:00"));
{
    Print("Events to run = ", EventNames, "<br>");
    Print("Start time = ", event\datetime, "<br>");
    Print("Repeat = ", event\repeat, "seconds", "<br>");
    Print("\timed.mml = ", event\script, "<br>");
}
```

Result

```
Event to run = event
Start time = 05-06-99 09:00:00
Repeat = 60 seconds
Script = \timed.mml
```

See also

`DelTimed`, `AddSession`

AddUser

AddUser creates a new account with directory space for files.

AddUser(account, domain, password, [force], [profile])



Only a verified account can use this function.

Parameters

Account

The name of the account.

Domain_name

The new account's domain.

Password

The user's password.

Force

If user exists, setting force to TRUE will over write all the users details. If it is a new user set to FALSE.

Profile

The name of the profile to apply to the user.

Returns

TRUE if account creation succeeds, otherwise FALSE.

Remarks

Accounts are created without features (POP3 account, forward etc.). For the account to start operating, the script must set the TYPE variable for the account using the appropriate constants OR'ed together.

Do not use this function to create SAM accounts.

Once the entry is created, e-mail can arrive for the new account. If the licence limits are exceeded, no account is created.

Example

This example shows how to create a new account with a mailbox and assign it to the Domain Base Profile for the first domain on the system:

```
Domain = "GMS.dom";
NewUser = "joe";

if (!ExistUser(#domain#\#NewUser#))
{
    if (AddUser(NewUser, Domain, "password", false, 1))
```



```
{
    #domain#\#newuser#\Type = BitSet(#domain#\#newuser#\Type,AccountTypeMailbox);
}
else
{
    Print(" Could not create new account.<br>");
}
}
else
{
    Print(" Account already exists!<br>");
}
```

This example checks if a user account exists:

```
Domain = "GMS.dom";
NewUser = "joe";
```

```
if (ExistUser(Resolve("#domain#\#NewUser#")))
{
    Print(" Account Exists.<br>");
}
else
{
    Print(" Account does not exist.<br>");
}
```

See also

AddDomain, AddSession, DelUser, DelUserFiles, ExistUser

AddZip

AddZip adds one or more files to a ZIP archive.

AddZip(zipfile,files[,separator][,absolutepath])



Only a verified account can add files with absolute paths.

Parameters

Zipfile

The ZIP archive to add the file to.

Files

A list of files to be added to the ZIP archive.

Separator (optional)

The character which separates files in the list, if this is not a space.

Absolutepath (optional)

If set to TRUE, this lets a verified account use absolute paths.

Returns

TRUE if the file is added successfully, otherwise FALSE.

Remarks

You cannot use MML to do things like extract files from a Zip file.

Example

```
zip = OpenZip("\\\\Test.zip");
if (zip)
{
    if (!AddZip(zip, "\\\\Timed.txt"))
    {
        Print("Couldn't add to zip");
    }
    CloseZip(zip);
    Print("Opened zip.");
}
else
{
    Print("Failed to open zip");
}
```

See also

CloseZip, OpenZip

AllowAbsoluteFileNames

AllowAbsoluteFileNames returns the status of the AllowAbsoluteFileNames global flag.

```
n = AllowAbsoluteFileNames()
```

Parameters

None.

Returns

TRUE if the global flag allowing absolute filenames is enabled, otherwise FALSE.

Example

```
print(allowabsolute filenames());
```

Result

0

ArchiveAddMessage

ArchiveAddMessage adds the specified message to the message archive.

ArchiveAddMessage(filename[,allowabsolute])

Parameters

filename

the filename of the message to be added.

allowabsolute(optional)

if set to true allows the use of absolute filenames.

Returns

Nothing

Example

```
if (!ArchiveAddMessage("CompanyA.dom\\listname\\archive\\vvfaaaaa.mbx"))
{
    print("Failed to add message<br>");
}
else
{
    print("Message added to archive<br>");
}

if (!ArchiveRefresh("CompanyA.dom\\listname"))
{
    print("Archive index was not refreshed<br>");
}
else
{
    print("Archive index was refreshed<br>");
}

if (!ArchiveRecalculate("CompanyA.dom\\listname"))
{
    print("Archive index was not recalculated");
}
else
{
    print("Archive index was recalculated");
}
```

Result

Adds a message to the archive then refreshes and recalculates the archive files.

See also

ArchiveRecalculate, ArchiveRefresh

ArchiveRecalculate

ArchiveRecalculate rebuilds the archive files for the given archive.

ArchiveRecalculate(archive)

Parameters

archive

the name of the archive to be recalculated.

Returns

Nothing

Example

```
if (!ArchiveAddMessage("CompanyA.dom\\listname\\archivein\\wvfaaaaa.mbx"))
{
    print("Failed to add message<br>");
}
else
{
    print("Message added to archive<br>");
}

if (!ArchiveRefresh("CompanyA.dom\\listname"))
{
    print("Archive index was not refreshed<br>");
}
else
{
    print("Archive index was refreshed<br>");
}

if (!ArchiveRecalculate("CompanyA.dom\\listname"))
{
    print("Archive index was not recalculated");
}
else
{
    print("Archive index was recalculated");
}
```

Result

Adds a message to the archive then refreshes and recalculates the archive files.

See also

ArchiveAddMessage, ArchiveRefresh

ArchiveRefresh

ArchiveRefresh refreshes the archive files for the given archive.

ArchiveRefresh(*archive*)

Parameters

archive

the name of the archive to be refreshed.

Returns

Nothing

Example

```
if (!ArchiveAddMessage("CompanyA.dom\\listname\\archivein\\wvfaaaaa.mbx"))
{
    print("Failed to add message<br>");
}
else
{
    print("Message added to archive<br>");
}

if (!ArchiveRefresh("CompanyA.dom\\listname"))
{
    print("Archive index was not refreshed<br>");
}
else
{
    print("Archive index was refreshed<br>");
}

if (!ArchiveRecalculate("CompanyA.dom\\listname"))
{
    print("Archive index was not recalculated");
}
else
{
    print("Archive index was recalculated");
}
```

Result

Adds a message to the archive then refreshes and recalculates the archive files.

See also

ArchiveAddMessage, ArchiveRecalculate

Asc

Asc returns the ASCII value of the first character in a string.

n = Asc(str)

Parameters

Str is a string.

Returns

- n — the ASCII value as an integer. Where str is a number, **Asc** returns the ASCII value of its most significant digit.
- 0 — if str is an empty string.

Example

```
Print (Asc("ABC"));
```

Result

65

See also

Chr

AutoConnect

AutoConnect tries to start a dial-up connection.

AutoConnect(RASPhoneBookEntry,RAS2PhoneBookRetry)

Parameters

RASPhoneBookEntry

Name of phone book entry. (Avoid spaces in names as they can cause problems on some NT systems.)

RAS2PhoneBookRetry

"Custom dialup", "Proxy custom dialup" or an empty string which means "Do not dial up".

"Custom dialup" reads these global Registry values:

- RASAccName, RASPassword and RASPhoneNumber.
- RASAccName2, RASPassword2 and RASPhoneNumber2.

"Proxy custom dialup" reads these global Registry values:

- ProxyRASAccName, ProxyRASPassword and ProxyRASPhoneNumber.
- ProxyRASAccName2, ProxyRASPassword2 and ProxyRASPhoneNumber2.

Returns

TRUE if the connection is made successfully, otherwise FALSE.

Remarks

If a connection is already available, the function uses that. A connection count is kept so that several services can use the same connection. You must use **AutoDisconnect** to close the connection.

If an application other than GMS established the connection, GMS still uses it, but it will not disconnect it. At shutdown NTMail/GMS closes any open connections.

Example

```
If (Autoconnect("Demon internet", "Custom dialup"))
{
    Print ("Connected");
}
```

Result

Connected

See also

AutoDisconnect, CollectFromPOP, IsConnected

AutoDisconnect

AutoDisconnect decrements the connection count and drops the current connection after the timeout period if the connection count has fallen to zero and the connection was established by GMS.

AutoDisconnect(timeout)

Parameters

Timeout is the number of 30 second ticks to wait before the connection is physically dropped.

Returns

TRUE if the disconnection succeeds, otherwise FALSE.

Remarks

The timeout is imposed because:

- HTTP makes lots of short-lived connections.
- The first three minutes of a connection cost most.

Example

This disconnects after two minutes:

```
if (AutoDisconnect(4))
{
    Print ("Disconnected");
}
```

See also

AutoConnect, IsConnected

BitAnd

BitAnd performs a bitwise AND on two integer numbers.

```
n = BitAnd(int1, int2)
```

Parameters

Bit

The first integer.

Int

The second integer.

Returns

The AND value as an integer.

Remarks

The least significant bit is bit 0.

Example

```
Print(BitAnd (12, 9));
```

Result

8

See also

BitIsSet, BitMaskIsSet, BitReset, BitSet

BitIsSet

BitIsSet tests whether bit position <bit> of the integer number <int> is set.

n = BitIsSet(int,bit)

Parameters

Int

The specified integer number.

Bit

The specified bit position within the integer.

Returns

TRUE if the specified bit is set, otherwise FALSE.

Remarks

The least significant bit is bit 0.

Example

This is how you test for a forwarding account:

```
User = "joe";  
Domain = "GMS.dom";  
  
if (BitIsSet(#domain#\#user#\Type,AccountTypeForward))  
{  
    Print("This is a forward account");  
}
```

Result

This is a forward account

See also

BitAnd, BitSet, BitMaskIsSet, BitReset

BitMaskIsSet

BitMaskIsSet tests whether all the bits in a mask are set in an integer number.

n = BitMaskIsSet(int,mask)

Parameters

Int

The specified integer number.

Mask

A bit mask specified as an integer.

Returns

TRUE if all the mask bits are set in the integer, otherwise FALSE.

Remarks

The least significant bit is bit 0.

Example

```
if (BitMaskIsSet (9, 13))
{
    Print("1 - All bits are set.",<br>);
}

if (BitMaskIsSet (13, 9))
{
    Print("2 - All bits are set.");
}
```

Result

1 - All bits are set.

See also

BitAnd, BitIsSet, BitSet, BitReset

BitReset

BitReset resets bit position <bit> of the number <int> and returns the result.

n = BitReset(int,bit)

Parameters

Int

The specified integer number.

Bit

The specified bit position within the number.

Returns

The integer with the reset bit.

Remarks

The least significant bit is bit 0.

Example

This example turns off the mailbox bit:

```
#domain#\#user#\Type = BitReset(#domain#\#user#\Type,AccountTypeMailbox);
```

See also

BitAnd, BitSet, BitIsSet, BitMaskIsSet, BitSet

BitSet

BitSet sets bit <bit> of number <int> and returns the result.

n = BitSet(int, bit)

Parameters

Int

The specified integer number.

Bit

The specified bit-position within the number.

Returns

An integer.

Remarks

The least significant bit is bit 0.

Example

This uses **BitSet** to test the account type:

```
if (!ExistUser(#domain#\#NewUser#))
{
    if (AddUser(NewUser, Domain, "password"))
    {
        #domain#\#newuser#\Type = AccountTypeMailbox;
        #domain#\#newuser#\Type =
            BitSet(#domain#\#newuser#\Type,AccountTypeForward);
        #domain#\#newuser#\Forward = ForwardEmail;
    }
}
```

See also

BitAnd, BitIsSet, BitMaskIsSet, BitReset

Bound

Bound makes sure that the result is between two specified bounds.

n = Bound(value, min, max)

Parameters

Value

The number to use.

Min

The minimum value for the variable.

Max

The maximum value for the variable.

Returns

If value is below "min", the function returns "min". If the value is above "max", it returns "max".

Example

```
x = Bound(12,5,8);  
Print ("x");
```

Result

8

See also

AddrInRange

CheckPassword

CheckPassword checks that the specified variable is the password. It does not matter if it is encrypted.

CheckPassword(account,uservar,password)

Parameters

Account

The account.

Uservar

The account variable to use. You can use this to check the various types of list password: joinpassword, leavepassword, etc.

Password

The number to use.

Returns

FALSE if the account does not exist or the password is wrong, otherwise TRUE.

Example

```
if (CheckPassword("companyA.dom\\joe",joinpassword,user-pwd))
{
    Print("Join password OK");
}
```

CheckServer

CheckServer resolves the server name and tries to connect to it on the specified port. Use it to test whether a service is running on the port.

CheckServer(server,port)

Parameters

Server

The server name.

Port

The port number to use.

Returns

TRUE if the connection succeeds, otherwise FALSE.

Remarks

After connecting to the server, the function disconnects.

Example

```
if (CheckServer("mail.companyA.dom",25))
{
    Print("Mail is up.");
}
```

Result

If mail is running on port 25 on the server, you will see:

Mail is up.

See also

ServiceStatus

CheckServiceAccess

CheckServiceAccess checks to see if the NT service control manager is accessible.

CheckServiceAccess()

Parameters

None

Returns

TRUE if access is available, otherwise FALSE.

Example

```
x = checkserviceaccess();  
print(x);
```

Result

1

CheckTopLevelScript

CheckTopLevelScript returns true if the current script is the top level entry script.

CheckTopLevelScript()

Parameters

None

Returns

TRUE if the current script is the top level entry script, otherwise FALSE.

Example

```
x = checkTopLevelScript();  
print(x);
```

Result

1

See also

KillScript, ListRunningScripts, SetScriptPriority

Chr

Chr returns a string containing the single character whose ASCII value is the specified integer.

```
str = Chr(int)
```

Parameters

Int is an integer number.

Returns

A one character string.

Example

```
x = Chr(65);  
Print ("x");
```

Result

A

See also

Asc

CloseMemberDB

CloseMemberDB closes a member database file.

CloseMemberDB(db_handle)

Parameters

Db_handle is the handle of the database to close, previously opened using **OpenMemberDb**.

Returns

Nothing.

Remarks

Once this function has been called, the database handle is destroyed so can no longer be used.

Example

This example opens a member database and prints the member field of all its records:

```
mydb = OpenMemberDB("test.com\\testlist",FILE_READ);

if (mydb)
{
    while (mymember = ReadNextMemberRecord(mydb))
    {
        Print("Account: ",mymember\member,"<BR>");
    }

    CloseMemberDB(mydb);
}
```

See also

GetAllMembers, GetMemberRecord, OpenMemberDb,
ReadNextMemberRecord, SetMemberRecord

CloseZip

CloseZip closes a Zip file.

CloseZip(handle)

Parameters

Handle is the file handle obtained using **OpenZip**.

Returns

TRUE if the file is closed successfully, otherwise FALSE.

Remarks

You cannot use MML to do things like extract files from a Zip file.

Example

```
zip = OpenZip("\\\\Test.zip");
if (zip)
{
    if (!AddZip(zip, "\\Timed.txt"))
    {
        Print("Couldn't add to zip");
    }
    CloseZip(zip);
    Print("Opened zip.");
}
else
{
    Print("Failed to open zip");
}
```

See also

AddZip, OpenZip

CollectFromPOP

CollectFromPop collects messages from a POP account for local delivery.

CollectFromPop
(**mailserver,port,account,password,deliveryclause,deleteoption,deliveryoption,addresslist,defaultaddress,deliveryrule,onlynewmessages,headerclause,errormsg**)

Parameters

Mailserver

The server to collect e-mail from.

Port

The port to connect to. This should be the POP port (110).

Account

The account name.

Password

The account password.

Deliveryclause

The clause to use if DeliveryRule is set to HeaderClause (see below).

DeleteOption

The action to take after reading the mail, one of these:

- 0 — do nothing.
- 1 — delete as download.
- 2 — delete only if rule matches.

DeliveryOption

AllAddressees, FirstAddressee or "To".

AddressList

A list of addresses.

DefaultAddress

If no rule matches, deliver to this address.

DeliveryRule

You can mix the DeliveryOption and DeliveryRule, for example to specify all addresses which match a rule, one of these:

- "GMSPOP3Domain" — delivery clause is empty.
- "HeaderClause" — delivery clause is the clause to look for in the message header. This can include wildcards, for example, "To: *@domain.dom".

- "EmailAddress" — the space-separated list of addresses used with a DeliveryOption of AllAddressees or FirstAddressee (see above).

OnlyNewMessages

Only download messages that have not already been read. This only works if the POP server supports the **top** command and provides "status:read" as the first line.

HeaderClause

The header clause you want the server to add to the messages. For example "X-FromPOP". If you don't want a header to be added use " ".

ErrorMsg

The variable into which the server will put any error message associated with the download.

Returns

TRUE if the function succeeds, otherwise FALSE.

Remarks

This function assumes that the TCP/IP connection to the mail server is available.

All e-mail goes through post@SMTP so that GMS Anti-Spam, Virus Scanner, etc. can be applied to it.

You can deliver e-mail to accounts on other machines too.

Example

The main parameter differences in the two examples are the following:

Parameter	First example	Second example
Deliveryclause	"To"	" "
DeliveryOption	"AllAddressees"	"To"
AddressList	" "	"bob joe lou"
DeliveryRule	"HeaderClause"	" "

This call uses a "To" HeaderClause to deliver to all addresses:

```
CollectFromPop("mail.companyA.dom",110,"guest","wh1sky",
"To",TRUE,"Alladdressees"," ","postmaster@companyA.dom",
"HeaderClause",FALSE,"X-FromPOP",errorMsg);
```

This call uses a "To" DeliveryOption to deliver to a list of addresses:

```
CollectFromPop("mail.CompanyA.dom",110,"guest","wh1sky",
```

```
" ",TRUE, "To","bob lou joe","postmaster@companyA.dom",  
"HeaderClause",FALSE,"X-FromPOP",errorMsg)
```

See also

AutoConnect, AutoDisconnect

ConfiguredLanguages

ConfiguredLanguages lists the short names for supported languages.

x = ConfiguredLanguages()

Parameters

None

Returns

A space separated list of short names for supported languages

Example

```
print(configuredlanguages());
```

Result

de en-us es es-mx fr it ja ko pl pt-br sv zh-tw

See also

SetSessionLanguage, LanguageName, DefaultLanguage, NLS,

ConvertCase

ConvertCase converts a string to all Lower or Upper case characters

ConvertCase(str,[true or false])

Parameters

Str

Str is a string

True

converts the string to Lowercase

False

converts the string to Uppercase

Returns

Case converted string

Example

```
str = "This is a Test";  
Quote = Convertcase(str,true);  
Print (Quote);
```

Result

this is a test

ConvertForDisplay

ConvertForDisplay converts an input string into the appropriate form for display within both HTML and plain text alternative sections of a multi-part list message.

x = ConvertForDisplay(string)

Parameters

String is the string for conversion into both an HTML statement and a plain text string.

Returns

Both an HTML statement and a plain text string for inclusion into the alternative sections of a multi-part message.

Remarks

All special characters are converted to their "&" version in the HTML portion of the text that is returned. For example, ">" is used for a ">" sign. In the plain text alternative section they are left unchanged.

Example

```
ConvertForDisplay(" £100 > £50 ");
```

Result

£100 > £50 (for use in the HTML section)
£100 > £50 (for use in the plain text section)

See also

ConvertToHTML

ConvertToAccount

ConvertToAccount converts a standard e-mail address into a "domain\user" account name.

user = ConvertToAccount(str[,domain])

Parameters

Str

The e-mail address string.

Domain (optional)

A string specifying the domain.

Returns

The address as "domain\user".

Remarks

- If the optional domain parameter is not specified, the system uses the domain name for the IP address this connection was made on.
- This function uses the GMS rules to strip any extraneous characters from the input.

Example 1

This example shows how the function converts many different address formats:

```
Print("Test.dom\Joe = ",
      ConvertToAccount("Joe","Test.dom"), "<br>");
Print("Test.dom\Joe = ",
      ConvertToAccount("Joe@Test.dom","companyB.dom"), "<br>");
Print("Test.dom\Joe = ",
      ConvertToAccount("From: <Joe> Joe Egg",Test.dom"), "<br>");
Print("Test.dom\Joe = ",ConvertToAccount("Test.dom!Joe",0), "<br>");
```

Result

```
Test.dom\Joe = Joe@Test.dom
Test.dom\Joe = Joe@Test.dom
Test.dom\Joe = Joe@Test.dom
Test.dom\Joe = Joe@Test.dom
```

Example 2

This example shows how to use this function on a logon page to automatically detect a domain if the user does not specify one:

```
FullUser = ConvertToAccount(User);
if (ExistUser(FullUser))
{
    if(CheckPassword(#FullUser#,password)
    {
        Print("Welcome.<br>");
    }
    else
    {
        Print("Logon failed. Please try again.<br>");
    }
}
else
{
    Print("Logon failed. Please try again.<br>");
}
```

See also

ConvertToEmailAddress, GetLocalDomain, ConvertToDomain

ConvertToDomain

ConvertToDomain finds the domain name in a string.

x = ConvertToDomain(account)

Parameters

Account is an account name.

Returns

The domain name.

Example

```
Print(ConvertToDomain("\"Jo Jones\" <jo@companyA.dom>"), "<br>");
```

Result

companyA.dom

See also

ConvertToEmailAddress, GetLocalDomain

ConvertToEmailAddress

ConvertToEmailAddress converts an account name into a standard e-mail address.

```
user = ConvertToEmailAddress(account)
```

Parameters

Account is an account name in any of the permitted account name forms.

Returns

A standard e-mail address in the form "user@domain".

Remarks

If no domain name is found, it appends the domain name of the current connection.

Example

```
ConvertToEmailAddress("companyA.dom\\joe");
```

Result

```
joe@companyA.dom
```

See also

ConvertToAccount, ConvertToDomain

ConvertToFlatHTML

ConvertToFlatHTML removes executable HTML entries (script, applet, embed, object) from supplied html

x = ConvertToFlatHTML(html)

Parameters

html

The html to be stripped of executable content.

Returns

Returns html code with executable content removed.

Example

```
messagetext = ConvertToFlatHTML(CurrentMsgText);
```

See also

ConvertToHTML

ConvertToHTML

ConvertToHTML converts an input string into an HTML statement.

x = ConvertToHTML(string)

Parameters

String is the string for conversion into an HTML statement.

Returns

An HTML statement.

Remarks

All special characters are converted to their "&" version in the text that is returned. For example, ">" is used for a ">" sign.

Example

```
ConvertToHTML("£100 > £50");
```

Result

```
&#163;100 &gt; &#163;50
```

See also

URLDecode, URLEncode

ConvertToJava

ConvertToJava converts supplied text to Java (escapes '"', '\', '\r', '\n')

```
x = ConvertToJava(string)
```

Parameters

String

The string to be converted to Java

Returns

Java.

Example

```
messagetext = ConvertToJava(messagetext);
```

ConvertToRealName

ConvertToRealName(string)

Parameters

String

is the string in RFC1522 format for conversion into a displayable format.

Returns

Converted string.

Remarks

used in header fields for multibyte/foreign names.

Example

```
address = ConvertToRealName(address);
```

ConvertToUser

ConvertToUser converts an MML user to a real user.

x = ConvertToUser(name)

Parameters

Name

is the MML user to be converted.

Returns

Username

Example

```
toAddress = ConvertToEmailAddress((ConvertToUser(ListEmailAddress) & "-join "),  
ConvertToDomain(ListEmailAddress));
```

Result

listname-join@domain.dom

See also

ConvertToEmailAddress, ConvertToDomain

ConvertToTime

ConvertToTime converts an input string into a standard time format.

```
t = ConvertToTime(string)
```

Parameters

String is a time string.

Returns

The time in hh:mm:ss format.

Example

```
ConvertToTime(" 19:00 ");
```

Result

```
19:00:00
```

See also

DateTimeFormat, Time

CreateDirectory

CreateDirectory creates a directory.

CreateDirectory(directory[,absolutepath])



Only a verified account can use this function if the folder location is outside of the Gordano file structure .

Parameters

Directory

The directory to create.

Absolutepath (optional)

If set to TRUE, this lets a verified account use absolute paths.

Returns

TRUE if the directory is created successfully, otherwise FALSE.

Example

This creates a directory in Joe's user space:

```
CreateDirectory("companyA.dom\\joe\\menus");
```

See also

DelDir

CreateSetup

CreateSetup creates the file Setup.txt in GMS's base directory.

CreateSetup(Flag)



Only a verified account can use this function.

Parameters

If Flag is set to FALSE, the function creates the plain file Setup.txt. If it's set to TRUE, the function creates a Setup.txt which includes other configuration files, such as postservers.txt and list help files.

Returns

TRUE if the file is created successfully, otherwise FALSE.

Remarks

You can do two things with the Setup.txt file this creates:

- Save it somewhere in case it's needed (it's essentially a backup).
- E-mail it to support or off site for safe storage.

Example

```
CreateSetup(TRUE);
Msg = MsgCreate("dean@company.dom", "fred@companyA.dom", "Setup");
if (msg)
{
    MsgAddFile(msg, "\\\\setup.txt", FILE_BINARY);
    MsgClose(Msg, MSG_SEND);
}
```

Date

Date returns the current date in standard format. Use it to create a date for use elsewhere.

t = Date([year, month, day])

Parameters

If the optional year, month and day parameters are specified, the function creates a date using these values.

Returns

The formatted date.

Remarks

The time is assumed to be 00:00.00.

Example

```
Print("The date is " , Date());
```

Result

The date is 1997-06-24

See also

Time, DateTimeFormat

DateTimeFormat

DateTimeFormat takes a date structure and prints it in the format requested.

t = DateTimeFormat(format_string, date)

Parameters

Format_string

One of the following variables:

String	Values	Description
yy	0-99	The last two digits of the year.
yyyy	0-9999	The year in full.
mm	1-12	The number of the month.
mmm	Jan-Dec	The first three letters of the month.
mmmm	January-December	The month name in full.
dd	1-31	The day of the month.
th	st,nd,rd,th	The appropriate letters for the given day of the month. For the 1st, this is "st".
dw	Mon-Sun	The current day of the week.
df	Monday-Sunday	The current day of the week in full.
hh	00-24	The hours.
mi	00-59	The minutes.
ss	00-59	The seconds.
r		Display in RFC format

Date

The date structure.

Remarks

You can use a "\" as an escape character.

Returns

The date string in the specified format.

Example

```
Print(DateTimeFormat(" ddth mmm yyyy hh:mi:ss", " 1997-02-06 12:02:22"));
```

Result

6th Feb 1997 12:02:22

See also

Date, Time

DefaultLanguage

DefaultLanguage returns the default system language

```
x = DefaultLanguage()
```

Parameters

None

Returns

the short name for the default system language. by default this is en-us.

Example

```
print(DefaultLanguage());
```

Result

en-us

DefaultListParm

DefaultListParm returns the default value of the specified list account parameter.

```
x = DefaultListParm(parameter)
```

Parameters

Parameter is a list account parameter.

Returns

A string containing the default value.

Example

```
x = DefaultListParm("maxtoaddresses");  
Print ("Maximum To addresses allowed = ", "x");
```

Result

Maximum To addresses allowed = 5

DelAlias

DelAlias removes an account's alias.

DelAlias(account,alias_name)



Only a verified account can use this function.

Parameters

Account

The account.

Alias_name

The alias to remove.

Returns

TRUE if the alias is successfully removed, otherwise FALSE.

Example

This removes the alias "Sales-team" from Joe's aliases:

```
DelAlias("companyA.dom\users\joe","Sales-team");
```

See also

AddAlias

DelDir

DelDir deletes a directory.

DelDir(directory[,absolutePath])



Only a verified account can use this function.

Parameters

Directory

The directory to remove.

Absolutepath (optional)

If set to TRUE, this lets a verified account use absolute paths.

Returns

TRUE if the directory is successfully removed, otherwise FALSE.

Example

This deletes a directory from Joe's user space:

```
DelDir("c:\users\joe\menus",TRUE);
```

See also

CreateDir

DelDomain

Removes a domain from the system.

DelDomain(domain_name)



Only a verified account can use this function.

Parameters

Domain_name is the domain to be removed.

Remarks

It may take up to 30 seconds to before the domain is deleted.

All details of the domain are lost, so to remove a domain cleanly, use the following calls in this order:

1. **DelUserFiles** and **DelUser** for all users.
2. **DelDomainFiles**.
3. **DelDomain** (once all files, directories and accounts have gone).

Similarly, do not delete a virtual domain's base domain without removing the virtual domain first.

Returns

TRUE if the domain is successfully removed, otherwise FALSE.

Example

```
DelDomain("companyA.dom");
```

See also

DelDomainFiles

DelDomainFiles

DelDomainFiles removes a domain's files from the system.

DelDomainFiles(domain_name)



Only a verified account can use this function.

Parameters

Domain_name is the domain whose files are to be removed.

Remarks

Use this function before **DelDomain**. The files removed include logs, templates ,etc. but not user files.

Returns

TRUE if the domain files are successfully removed, otherwise FALSE.

Example

```
DelDomainFiles("companyA.dom");  
DelDomain("companyA.dom");
```

See also

DelDir, DelDomain, DelUserFiles

DeleteMemberRecord

DeleteMemberRecord deletes a member record from the database.

DeleteMemberRecord(handle)

Parameters

Handle is the handle for the record created by **GetMemberRecord**.

Returns

TRUE if the record is successfully removed, otherwise FALSE.

Example

This deletes the first five records from the database:

```
dbase = OpenMemberDb(companyA.dom\stafflist,FILE_WRITE);
if (dbase)
{
    for (i = 0; i < 5; i = i + 1)
    {
        DeleteMemberRecord(dbase);
    }
}
CloseMemberDb(dbase);
```

See also

GetMemberRecord, ReadNextMemberRecord, SetMemberRecord

DelFile

DelFile deletes a file.

DelFile(filename[,absolutepath])



Only a verified account can use this function.

Parameters

Filename

The name of the file to delete.

Absolutepath (optional)

If set to TRUE, this lets a verified account use absolute paths.

Returns

TRUE if the file is removed, otherwise FALSE.

Example

```
if (DelFile(note.txt))  
{  
    Print("Note.txt deleted");  
}
```

See also

DelUserFiles, DelDomainFiles, ExistFile, FileReplace, FileOpen, FileClose

DelSession

DelSession deletes the current session, effectively logging off the user.

DelSession()

Parameters

None.

Returns

Nothing.

Remarks

If the user was a verified account, they become unverified. All global variables for the account are lost.

Example

```
DelSession();
```

Result

None.

See also

AddSession

DelTimed

DelTimed removes an event from the schedule list.

DelTimed(event)



Only a verified account can use this function.

Parameters

Event is the name of the event to delete.

Returns

TRUE if the event is removed, otherwise FALSE.

Remarks

If the event is currently in progress, it will not be deleted until it ends.

Example

```
AddTimed("event", "05-06-99", 60, "\timed.mml", "0123456", "09:00:00", "17:30:00");
{
    Print("Events to run = ", EventNames, "<br>");
    Print("Start time = ", event\datetime, "<br>");
    Print("Repeat = ", event\repeat, "seconds", "<br>");
    Print("\timed.mml = ", event\script, "<br>");
}
DelTimed(event);
```

See also

AddTimed

DelUser

DelUser deletes the account from the system.

DelUser(account)



Only a verified account can use this function.

Parameters

Account is the name of the account to delete.

Returns

TRUE if the account is successfully deleted, otherwise FALSE.

Remarks

You must delete the account's files first using **DelUserFiles**.

If **DelUser** succeeds, all information about the account is removed — to re-create the user, you must create the account again from scratch.

Example

```
user = "companyA.dom\joe";
if (DelUserFiles(user))
{
    if (DelUser(user))
    {
        Print("Removed user ",ConvertToEmailAddress(user));
    }
    else
    {
        Print("Could not delete ",ConvertToEmailAddress(user));
    }
}
```

Result

Could not delete user joe@companyA.dom
Error result 15 = No such user name.

See also

DelUserFiles

DelUserFiles

DeleteUserFiles removes an account's files.

DelUserFiles(account)



Only a verified account can use this function.

Parameters

Account is the name of the account to delete.

Returns

TRUE if the files are successfully deleted, otherwise FALSE.

Remarks

Before you delete an account, use this function to delete its files.

Example

```
user = "companyA.dom\\joe";
if (DelUserFiles(user))
{
    if (DelUser(user))
    {
        Print("Removed user ",ConvertToEmailAddress(user));
    }
    else
    {
        Print("Could not delete ",ConvertToEmailAddress(user));
    }
}
```

See also

DeleteUser

DiffDate

DiffDate gives the difference between two date and time variables.

n = DiffDate(date_1,date_2)

Parameters

Date_1

The first date.

Date_2

The second date.

Returns

The difference in seconds.

Remarks

You can use the **Date** function to specify the dates. Specify the most recent date/time first.

Example

```
Print(DiffDate(" 1997-06-24 08:00:00 "," 1997-08-21 07:00:00 "));  
Print("<br>");  
Print(DiffDate(Date(),Date(1999,12,31)),"<br>");
```

Result

```
5007600  
17753712
```

See also

Date, DateTimeFormat

DirSize

DirSize returns the size and number of files in each of the specified directory's subdirectories.

str = DirSize(dir[,absolute_path])

Parameters

Dir

The directory to return size information for.

Absolute_path (optional)

If set to TRUE, this lets a verified account use absolute paths.

Returns

A string listing the subdirectory names, file sizes and no of files in the directory. The subdirectory, size and file count entries are separated by a semicolon. The last two semi-colon separated entries are the size and number of files for the base directory specified in DirSize().

Example

```
Print(DirSize("companyA.dom\\\\"));
```

Result

```
\\BadMes;0;0 \\MesLog;0;0 \\Users\\List;4543;1 \\Users\\PostMaster;0;0 \\Users;4543;1 ;4579;2
```

EncryptPassword

EncryptPassword encrypts a password

```
str = EncryptPassword(password)
```

Parameters

Password is the password to encrypt.

Returns

The password as an encrypted string.

Example

```
code_pwd = EncryptPassword(pwd);  
Print (code_pwd);
```

Result

```
dsjfg9803skj
```

See also

VerifyUser

EnumRasEntries

EnumRasEntries lists RAS entries from your phone book in URL-encoded form.

```
str = EnumRasEntries()
```

Parameters

None.

Returns

A space-separated list of phone book entries, each URL-encoded.

Remarks

RAS is the Remote Access Service.

Example

```
Print(EnumRasEntries());
```

Result

Demon+Internet INS+Net+++ISDN INS+Net

See also

AutoConnect

ExistDomain

ExistDomain checks to see if the domain specified exists on this server. That is, it searches for it in the Registry.

ExistDomain(domain)

Parameters

Domain is the name of the domain to look for.

Returns

TRUE if the domain exists, otherwise FALSE.

Example

This uses **ExistDomain** to check that the domain exists before trying to delete it:

```
dom = "companyA.dom";  
if (ExistDomain(dom))  
{  
    DelDomainFiles(dom);  
    DelDomain(dom);  
}
```

See also

AddDomain, ExistUser

ExistFile

ExistFile tests whether a file exists.

ExistFile(filename[,absolutePath])

Parameters

Filename

The name of the file to look for.

AbsolutePath (optional)

If set to TRUE, this lets a verified account use absolute paths.

Returns

TRUE if the file exists, otherwise FALSE.

Example

This uses **ExistFile** to check that a file exists before trying to delete it:

```
textfile = "companyA.dom";  
if (ExistFile(textfile))  
{  
    DelFile(textfile);  
}
```

See also

ExistDomain

ExistUser

ExistUser tests whether an account exists, using the same routines as GMS to try to resolve its name.

ExistUser(account [,domain])

Parameters

Account

The user name in standard form. That is, separated by "@", "\", or "!".

Domain (optional)

The domain name to add to the account name if the domain name is not specified. If the domain name is not specified, it uses the domain name associated with the connection.

Returns

TRUE if the account is on this system, FALSE if the function could not interpret the address or the account does not exist.

Example

```
if (!ExistUser(domain & "\\" & newuser))
{
    if (AddUser(NewUser,Domain,"password"))
    {
        #domain#\#newuser#\Type =
        BitSet(#domain#\#newuser#\Type,AccountTypeMailbox);

        Print ("Account created");
    }
}
else
{
    Print ("Account already exists");
}
```

See also

AddUser, DelUser, BitSet

ExistVar

ExistVar checks through all known variables to see if the supplied variable name exists for this session.

ExistVar(varname)

Parameters

Varname is the name of the variable to look for.

Returns

TRUE if a variable with the specified name exists, otherwise FALSE.

Remarks

All variable types are checked — system, global, connection, constants etc.

Example

```
if (ExistVar("LOG_STATS"))
{
    Print("Log stats exist");
}
```

FileClose

FileClose closes the file, writes out all buffers and releases all resources associated with the file handle.

FileClose(handle)

Parameters

Handle is the file handle created by **FileOpen**.

Returns

TRUE if the file closed successfully, FALSE if the file has already been closed or the close failed.

If the handle supplied is not a file handle, the script interpreter reports an error.

Remarks

If an error occurs during a script and file handles are open, they are automatically closed.

Example

```
file = FileOpen(filename);
if (file)
{
    while (!FileEof(file))
    {
        Print(FileReadLine(file), "<br>\r\n");
    }
}
FileClose(file);
```

See also

FileOpen, FileReadLine, FileWriteLine, FolderOpen, DelFile

FileCopy

FileCopy Copies a file from one location to another.

FileCopy(from,to[,overwrite[,absolute]])

Parameters

from

location and filename of file to be copied

to

location and filename the file is to be copied to

overwrite

if false the copy will fail if the file specified in "to" already exists. By default overwrite is true.

absolute

If set to true and the user is verified fully specified paths can be used.

Example

```
if (!FileCopy("\\\\setup.txt", "D:\backup\setup.txt", TRUE, TRUE))
{
    WarningMsg = "Couldn't save setup.txt file to " & destfile;
}
```

FileEOF

FileEOF checks whether the end of a file opened with read access has been reached.

tf = FileEOF(handle)

Parameters

Handle is the file handle created using **FileOpen**.

Returns

TRUE if the end of the file has been reached, otherwise FALSE.

Example

```
file = FileOpen(filename);
if (file)
{
    while (!FileEof(file))
    {
        Print(FileReadLine(file), "<br>\r\n");
    }
}
FileClose(file);
```

See also

FileReadLine, FileWriteLine

FileIsBinary

FileIsBinary determines whether a file is binary (contains nulls or chars > 127)

FileIsBinary(filename)

Parameters

filename

the file to be checked

Returns

True if the file is binary else false.

Example

```
if fileisbinary("\\\\setup.txt")
{
    WarningMsg = "File is Binary";
}
```

Filemd5

Filemd5 returns a string representing the MD5 digest of the specified file.

Filemd5(filename[,allowabsolute])

Parameters

filename

the file to be checked

allowabsolute

if set to true allows a verified user to specify an absolute path to the filename.

Returns

a string representing the MD5 digest of the specified file

Example

```
x = filemd5("CompanyA.dom\\joe\\attachments\\file.txt");  
print(x);
```

Result

7a81ffb7c7670cda4eeb068d57ff629b

See Also

Md5Str

FileOpen

FileOpen opens a file.

x = FileOpen(filename,type[,absolutepath])



Only a verified account can use this function for writing and with absolute file names.

Parameters

Filename

The file to open, in the form "domain\user\filename".

Type

The operation to perform on the file: FILE_READ, FILE_WRITE or FILE_APPEND.

Absolutepath (optional)

If set to TRUE, this lets a verified account use absolute paths.

Returns

- FALSE — failed to open the file as requested, for example due to a sharing violation.
- X — the file handle to be used in other file operations.

Remarks

The function **Resolve** may be useful in creating a filename. Once the file handle has been opened, the variable id/filename is available. This contains the name of the file that is currently being accessed.

Example

```
filename = #dir#\plan;  
file = FileOpen(Resolve("#dir#\#filename#"),FILE_READ);  
if (file)  
{  
    while (!FileEof(file))  
    {  
        Print(FileReadLine(file),"<br>\n");  
    }  
    FileClose(file);  
}
```

Result

The file is opened.

See also

FileClose, FileEOF, FileReadLine, FileWriteLine, Resolve

FileReadLine

FileReadLine reads a line from a file opened for read operations and returns the whole line (without its CRLF).

```
str = FileReadLine(handle)
```

Parameters

Handle is the file handle returned by **FileOpen**.

Returns

If End-of-File is found, an empty string is returned.

Remarks

Checking for an empty string is not enough to determine whether the end of a file has been reached.

Example

This example prints all the lines of a file:

```
while (!FileEof(filehandle))
{
    line = FileReadLine(filehandle);
    Print(line);
}
FileClose(filehandle);
```

See also

FileClose, FileEOF, FileReadLine, FileWriteLine

FileReplace

FileReplace replaces one file with another.

FileReplace(file_to_go,replacement[,absolutepath])



Only a verified account can use this function.

Parameters

File_to_go

The file that's to be replaced.

Replacement

The file that will replace file_to_go.

Absolutepath (optional)

If set to TRUE, this lets a verified account use absolute paths.

Returns

Nothing.

Example

This replaces the file timed.txt with its backup file timed.bak:

```
FileReplace("\\\\timed.txt", "\\\\timed.bak");
```

See also

DelFile

FileSize

FileSize returns the size of specified file

FileSize(filename[,absolutePath])

Parameters

filename

The file to return the size of.

Absolutepath (optional)

If set to TRUE, this lets a verified account use absolute paths.

Returns

The size of the specified file in bytes. Returns 0 on error.

Example

```
x = FileSize("jtest.dom\\j\\inbox.mbx");  
print(x);
```

See also

FileOpen, FileClose

FileVscan

FileVscan checks a file for viruses.

filevscan(filename,result[,absolutepath])



A valid virus scanning package needs to be installed and configured.

Parameters

Filename

The file that's to be scanned.

Result

String to return if virus is found.

Absolutepath (optional)

If set to TRUE, this lets a verified account use absolute paths.

Returns

True if the scan is performed, and result is set true if the file contains a virus.

Example

```
if (!FileVScan("C:\file.exe",infected,true))
{
    WarningMsg = "File was not scanned";
}
elseif (infected)
{
    WarningMsg = "Virus found in file";
}
```

FileWriteLine

FileWriteLine writes one or more lines to a file, automatically terminating each with a CRLF.

FileWriteLine(handle,string)



Only a verified account can use this function.

Parameters

Handle

The file handle returned by **FileOpen**.

String

The line to write to the file.

Returns

This returns an error if the file was opened read-only.

Remarks

If string has multiple lines the function writes more than one line to the file. It terminates these with CRLFs as required.

Example

This example writes 20 lines to a file:

```
filename = #dir#\plan;
file = FileOpen(Resolve("#dir#\#filename#"),FILE_WRITE);
if (file)
{
    for (i = 0; i < 20; i = i + 1)
    {
        FileWriteLine(file, "<br>\r\n");
    }
    FileClose(file);
}
```

See also

FileClose, FileEOF, FileReadLine

FilterDomainsOfType

FilterDomainsOfType selects domains of the specified type

```
str = FilterDomainsOfType(string,mask)
```

Parameters

String

A space-separated list of domains.

Mask

A mask of domain types. You can specify a single type using `DomainTypeFull`, `DomainTypePOP`, `DomainTypeRobot` or `DomainTypeVirtual`.

Returns

A space-separated list of all the domains of the given type.

Example

```
domains = FilterDomainsOfType("\\domain names",DomainTypeFull);  
Print(domains);
```

Result

```
Fullldomain1,Fullldomain2
```

See also

`FilterUsersOfType`

FilterMsg

FilterMsg searches a message for any of a number of strings.

```
str = FilterMsg(msg,"string1","string2",.....)
```

Parameters

Msg

The handle produced using **MsgCreate** or **MsgCopy**.

Mask

A comma-separated sequence of up to 63 strings.

Returns

For each filter, a count of how many occurrences were found.

Remarks

The strings must not contain wildcards.

Example

In this example Fred appears in the message five times and Joe eight times. The name Lou is not in the message:

```
msg = MsgCopy(msg_4);  
FilterMsg(msg,"Fred","Joe","Lou");
```

Result

5 8 0

See also

WildcardFilterMsg

FilterUsersOfType

FilterUsersOfType selects accounts of the specified type.

str = FilterUsersOfType(userlist,attribute,bitvar,separator)

Parameters

Userlist

A space-separated list of accounts.

Attribute

The Registry variable to search for. For example, you might search for accounts with AccessRights set to domain administrator. See the *GMS Reference Guide* for details of Registry variables.

Bitvar

Either a bit position or the name of the variable, for example the constant ACCESS_RIGHT_SYSTEM.

Seperator

The seperator to be used between each user returned.

Returns

A list of accounts.

Example

This example returns all accounts in UserAccessList who have domain administrator rights:

```
x = FilterUsersOfType(UserAccessList(), "AccessRights", ACCESS_RIGHT_SYSTEM, ":");  
Print(x);
```

Result

Domain-admin1:Administrator

See also

FilterDomainsOfType

FindFiles

FindFiles searches for the named files or directories.

```
str = FindFiles(filename[,timestamp[,absolute[,filetype]])
```

Parameters

Filename

A space-separated list of file names. This can include wildcards.

Timestamp (optional)

If set to TRUE, this retruns the timestamp for the files

Absolutepath (optional)

If set to TRUE, this lets a verified account use absolute paths.

Filetype (optional)

If set to TRUE, this returns a flag denoting a file or directory.

Returns

A space-separated list of filenames. Each entry shows:

Name;size in bytes;date modified or created;file or directory.

Remarks

If a directory appears in the list, its subdirectories are searched too. However, the subdirectory structure is not output. Directory entries are denoted by "d" when the filetype varibale is set.

Example

```
files = findfiles("c:\\gordano\\company.dom\\users\\user1\\*.mbx",1,1,1);  
Print(files);
```

Result

```
InBox.mbx;29021;2003-05-16;20:11:45;f
```

See also

ExistFile, FolderExist

FolderAppendMsg

FolderAppendMsg appends a message to a folder (mailbox).

x = FolderAppendMsg(foldername,msg)

Parameters

Foldername

The folder name.

Msg

The message to add.

Returns

The UDIL of the message (the id taken from its header).

Remarks

You must specify the folder name fully, as in the following example.

Example

This appends a message (n) from Lou's mailbox to Joe's mailbox:

```
folder1 = FolderOpen("companyA.dom\Lou\inbox");  
msg = MsgCopy(folder1\n);
```

```
foldername = "companyA.dom\joe\inbox"  
folder2 = FolderOpen(foldername);  
FolderAppendMsg(foldername,msg);
```

```
folderclose(folder1,FALSE);  
folderclose(folder2,TRUE);
```

See also

FolderClose, FolderList, FolderGetMessageCount,
FolderMsgCheckStatus, FolderMsgSetStatus, FolderOpen,
FolderExist, FolderGetNewMessageCount

FolderClose

FolderClose closes a folder (mailbox).

FolderClose(handle,save)

Parameters

Handle

The folder handle set by **FolderOpen**.

Save

If Save is TRUE, save any changes made to the folder.

Returns

Nothing.

Example

This appends a message (n) from Lou's mailbox to Joe's mailbox. It uses a Save value of TRUE for the folder which has been changed:

```
folder1 = FolderOpen("companyA.dom\Lou\inbox");  
msg = MsgCopy(folder1\n);
```

```
folder2 = FolderOpen("companyA.dom\Joe\inbox");  
FolderAppendMsg(folder2,msg);
```

```
folderclose(folder1,FALSE);  
folderclose(folder2,TRUE);
```

See also

FolderAppendMsg, FolderDelete, FolderList,
FolderMsgCheckStatus, FolderOpen, FolderExist

FolderDelete

FolderDelete deletes a folder (mailbox).

FolderDelete(name)

Parameters

Name is the name of the folder, excluding its ".mbx" extension.

Returns

TRUE if the folder is deleted successfully, otherwise FALSE.

Remarks

The .mbx file and .idx file are deleted

Example

```
FolderDelete("companyA.dom\\joe\\inbox");
```

See also

FolderAppendMsg, FolderClose, FolderList,
FolderGetMessageCount, FolderMsgCheckStatus, FolderOpen,
FolderExist, FolderGetNewMessageCount

FolderExist

FolderExist checks to see if the specified folder exists.

FolderExist(folder)

Parameters

Folder

is the name of the folder, excluding its ".mbx" extension.

Returns

TRUE if the folder exists, otherwise FALSE.

Example

```
if (!FolderExist("companyA.dom\joe\inbox"))  
{  
    print("Folder inbox does not exist");  
}
```

See also

FolderAppendMsg, FolderClose, FolderList,
FolderGetMessageCount, FolderMsgCheckStatus, FolderOpen,
FolderGetNewMessageCount

FolderFlush

FolderFlush removes messages marked for deletion from a folder.

FolderFlush(folder)

Parameters

Folder

is the name of the folder, excluding its ".mbx" extension.

Returns

False if the folder does not exist otherwise True.

Example

```
FolderFlush("companyA.dom\joe\inbox"))
```

See also

FolderAppendMsg, FolderClose, FolderList,
FolderGetMessageCount, FolderMsgCheckStatus, FolderOpen,
FolderGetNewMessageCount

FolderGetMessageCount

FolderGetMessageCount counts the number of messages in a folder (mailbox).

n = FolderGetMessageCount(folder)

Parameters

Folder is the name of the folder, excluding its ".mbx" extension.

Returns

The number of messages or 0 (zero) if the mailbox does not exist.

Example

```
n = FolderGetMessageCount("companyA.dom\\joe\\inbox");
if n > 128
{
    Print("folder has reached message limit");
}
```

See also

FolderAppendMsg, FolderList, FolderMsgCheckStatus,
FolderMsgSetStatus,
FolderMsgUnsetStatus, FolderGetNewMessageCount

FolderGetNewMessageCount

FolderGetNewMessageCount counts the number of new messages in a folder (mailbox).

n = FolderGetNewMessageCount(folder)

Parameters

Folder

is the name of the folder, excluding its ".mbx" extension.

Returns

The number of messages or 0 (zero) if the mailbox does not exist or there are no new messages.

Example

```
n = FolderGetNewMessageCount("companyA.dom\\joe\\inbox");
if n > 0
{
    Print("You have ",n," new messages");
}
```

See also

FolderAppendMsg, FolderList, FolderMsgCheckStatus,
FolderMsgSetStatus, FolderMsgUnsetStatus,
FolderGetMessageCount

FolderList

FolderList lists folders (mailboxes).

FolderList(folder_wildcard)

Parameters

Folder_wildcard is a wildcarded string of folder names.

Returns

A comma-separated list of folders (files with .mbx extension) and directories . Directories have "/" following their name.

Example

```
FolderList("companyA.dom\joe");
```

Result

AddressBooks/,Drafts,InBox,Sent Items,Sent,subscribed/,templates,Trash

See also

FolderAppendMsg, FolderClose, FolderDelete,
FolderMsgCheckStatus, FolderMsgSetStatus, FolderOpen

FolderModified

FolderModified lists folders (mailboxes).

FolderModified(folder)

Parameters

Folder

is the name of the folder, excluding its ".mbx" extension.

Returns

The modified time (as a date variable) for the specified folder.

Example

```
FolderModified("companyA.dom\joe\inbox");
```

Result

```
2001-04-19 08:44:10
```

See also

FolderAppendMsg, FolderClose, FolderDelete,
FolderMsgCheckStatus, FolderMsgSetStatus, FolderOpen

FolderMsgCheckStatus

FolderMsgCheckStatus checks whether the status of the message is as specified.

FolderMsgCheckStatus(handle,msg,status)

Parameters

Handle

The folder handle.

Msg

The number of the message in the mailbox.

Status

The status to check for. This is one of the "FOLDER_MSG_" constants listed in "Constants" on page 297.

Returns

There are two cases:

- If Status is FOLDER_MSG_ADVERT_POS, FOLDER_MSG_ADVERT_COUNT or FOLDER_MSG_ADVERT_MAX, the function returns the value of that status.
- If Status is anything else, the function returns TRUE/FALSE.

Example

```
folder = FolderOpen("companyA.dom\\#user#\\inbox");  
count = FolderMsgCheckStatus(folder,4,FOLDER_MSG_ADVERT_COUNT);
```

```
Print("Advert total is ",count," adverts.");
```

Result

Advert total is 6 adverts.

See also

FolderList, FolderGetMessageCount, FolderMsgSetStatus, FolderMsgUnsetStatus

FolderMsgSetStatus

FolderMsgSetStatus gives a message the specified status.

FolderMsgSetStatus(handle,msg,status[,value])

Parameters

Handle

The folder handle.

Msg

The number of the message in the mailbox.

Status

The status to give the message, one of the "FOLDER_MSG_" constants listed in "Constants" on page 297.

Value

If Status is FOLDER_MSG_ADVERT_POS, FOLDER_MSG_ADVERT_COUNT or FOLDER_MSG_ADVERT_MAX, this becomes its value.

Returns

Nothing.

Example

This sets the status of the fourth message in the folder to "posted":

```
folder = FolderOpen("companyA.dom\\joe\\inbox");  
FolderMsgSetStatus(folder,4,FOLDER_MSG_POSTED);
```

See also

FolderAppendMsg, FolderGetMessageCount,
FolderMsgCheckStatus, FolderMsgUnsetStatus

FolderMsgUnsetStatus

FolderMsgUnsetStatus removes the specified status value.

FolderMsgUnsetStatus(handle,msg,status)

Parameters

Handle

The folder handle.

Msg

The number of the message in the mailbox.

Status

The status to remove from the message. This is one of the "FOLDER_MSG_" constants listed in "Constants" on page 297.

Returns

Nothing.

Example

This removes the "posted" status from the fourth message in the folder:

```
folder = FolderOpen("companyA.dom\\joe\\inbox");  
FolderMsgUnsetStatus(folder,4,FOLDER_MSG_POSTED);
```

See also

FolderAppendMsg, FolderGetMessageCount,
FolderMsgCheckStatus, FolderMsgSetStatus

FolderOpen

FolderOpen opens the folder (mailbox) for use by other functions.

x = FolderOpen(name)

Parameters

The name of the folder to open.

Returns

If the folder is opened successfully, it returns a handle for use by other calls. If it fails it returns FALSE.

Remarks

If the folder does not exist, the function creates it.

Example

```
folder = FolderOpen("companyA.dom\\joe\\inbox");
{
    if(FolderMsgSetStatus(folder,4,FOLDER_MSG_POSTED))
    {
        Print("The message has been posted");
    }
}
```

See also

FolderClose, FolderDelete, FolderList, FolderGetMessageCount, FolderMsgCheckStatus

FolderRename

FolderRename renames a folder.

FolderRename(original,replacement)

Parameters

original

The current name and location of the folder

replacement

The name the folder is to be changed to.

Returns

True if the rename is successful else it returns false.

Example

FolderRename("CompanyA.dom\\joe\\trash","litter")

See also

FolderClose, FolderDelete, FolderList, FolderGetMessageCount,
FolderMsgCheckStatus, FolderGetNewMessageCount

GetAllMembers

GetAllMembers produces a list of list members.

```
str = GetAllMembers(format_string, account[,type])
```

Parameters

Format_string

Defines information required on each member, using a "% " to indicate the field to be completed. The case-sensitive fields are:

Field	Meaning
%a	Acknowledge
%A	AcceptCommand
%b	NumBadMessages
%c	DateConfirmSent
%d	Domain
%f	ForDays
%g	DigestType
%h	HelpSentTime
%i	DigestSet
%l	DigestTypeSet
%J	DateJoined
%j	TimeJoined
%L	DateLeft
%l	TimeLeft
%m	DateToCommandModerator
%n	Name (userid)
%p	MyPassword
%r	LeftReason
%s	Show
%S	ShowSet
%t	Type
%T	String version of Type
%u	Suspended
%U	UDIL
%X	DateExpiry
%x	TimeExpiry

To specify a user-defined field, type "\$ " followed by its name.

To mix text with a user-defined field, type "\$" followed by its name between "{" and "}" brackets. You can type any other text you want on either side of the brackets (but after the "\$"). The example below shows how to do this.

Account

The list account.

Type (optional)

The account type, as listed below. If no type is specified, the call returns all accounts.

Type	Lists
MEMBER_TYPE_MEMBER	All current members.
MEMBER_TYPE_OLD	Members who've left.
MEMBER_TYPE_BANNED	Banned members.
MEMBER_TYPE_BEING_MODERATED	Being moderated.
MEMBER_TYPE_CONFIRM	Awaiting confirmation.
MEMBER_TYPE_JOIN_EXPIRED	Expired members.
MEMBER_TYPE_DELETE	Deleted members.
MEMBER_TYPE_ALL	All members

Returns

A string of list members in the required format.

Remarks

This function automatically opens the database, obtains the results then closes the database. It is easier to use because it opens and closes the database for you.



To get more than one record using this function, get them in alphabetical order. The function cannot "rewind" back up the alphabet.

Example

```
colour = blue;
Print(GetAllMembers(" %n@d<BR> ", "zak.dom\\list"));
Print(GetAllMembers(" %n $colour<BR> ", "zak.dom\\list"));
Print(GetAllMembers("$old {colour} eyes<BR> ", "zak.dom\\list"));
```

Result

```
joe@companyA.dom<BR> david@domain.dom<BR>
joe blue<BR>
old blue eyes<BR>
```

See also

OpenMemberDb, GetMembeRecord, MemberFormat

GetConnectionDomain

GetConnectionDomain returns the local domain of the connection on the server.

str = GetConnectionDomain()

Parameters

None.

Returns

the local domain of the connection on the server

Example

```
x = GetConnectionDomain();  
print(x);
```

Result

companyA.dom

See also

GetLocalServerAddress, GetConnectionVariables

GetConnectionVariables

GetConnectionVariables lists a connection's parameters.

str = GetConnectionVariables()

Parameters

None.

Returns

A space-separated list of parameters associated with the connection, as shown in a URL.

Example

```
GetConnectionVariables();
```

Result

```
http_accept http_accept_encoding http_accept_language http_connection http_cookie  
http_host http_user_agent local_hostname remote_hostname request_script_name  
script_length script_name server_name server_port server_protocol server_software
```

See also

GetLocalServerAddress, GetConnectionDomain

GetHostedIps

GetHostedIps gets the IP addresses of all network cards on the server.

str = GetHostedIps()

Parameters

None.

Returns

A space-separated list of IP addresses.

Example

```
Print(GetHostedIps());
```

Result

192.54.12.78 192.54.12.84 192.54.12.112

See also

GetLocalIps

GetHostname

GetHostname performs a reverse lookup on the address specified.

```
str = GetHostname(IP_address)
```

Parameters

IP_address is the IP address to look up.

Returns

Name of the host.

Remarks

This is the opposite of **GetIPAddress**.

Example

```
Print (Gethostname(" 194.205.1.152 "));
```

Result

mail.companyA.dom

See also

GetIPAddress, GetMXRecord

GetHTTPCookie

```
GetHTTPCookie("GordanoSearchString");
```

Parameters

GordanoSearchString

The identifying string used when the cookie was set using SetHTTPCookie.

Remarks

Cookie needs to have been set on the client machine using **SetHTTPCookie** otherwise nothing will be returned.

Returns

Any cookie on the client machine with a matching *GordanoSearchString*. Specifically the content of the "data" parameter as set by SetHTTPCookie.

Example

```
signoninfo = GetHTTPCookie("GordanoSignon");
```

See also

SetHTTPCookie

GetHTTPPage

GetHTTPPage gets the HTTP page specified.

GetHTTPpage(URL[,filename[,contenttype[absolutePath]]]);

Parameters

URL

The HTTP page to be retrieved.

filename(optional)

filename to save the page to.

contenttype(optional)

specifies the name of a variable that will be set to the data
ContentType

absolutePath(optional)

if set to true allows a verified user to specify an absolute
path for filename.

Returns

The requested HTTP page

Example

```
URL = "http://www.CompanyA.dom/logon.htm?Logon=Y"
if (ExistVar("password"))
{
    URL = URL & "&password=" & URLEncode(Trim(password));
}
if (ExistVar("username"))
{
    URL = URL & "&username=" & URLEncode(Trim(username));
}
print(gethttppage(url));
```

Result

The page `http://www.companyA.dom/logon.htm?Logon=Y&password=ae^$&q&username=jyei^e%` is displayed in the browser window.

GetIPAddress

GetIPAddress performs a lookup on the given host and returns its IP address.

```
addr = GetIPAddress(hostname)
```

Parameters

Hostname is the name of the host whose A Name or C Name record is to be looked for.

Returns

The IP address of the host.

Remarks

This is the opposite of **GetHostname**.

Example

```
Print(GetIPAddress("mail.companyA.dom"));
```

Result

194.205.1.152

See also

GetHostname, GetLocalDomain, GetMXRecord

GetLoadsharingServer

GetLoadsharingServer shows which server in a load sharing array handles the account.

```
addr = GetLoadsharingServer(account)
```

Parameters

Account is the account whose server you want.

Returns

The IP address of the server in the load sharing array which handles the account.

Example

```
server = GetLoadsharingServer(dean@companyA.dom);  
Print(server);
```

Result

```
server1
```

See also

GetLoadsharingServerList

GetLoadsharingServerList

GetLoadsharingServerList lists the servers in the load sharing array.

str = GetLoadsharingServer(domain)

Parameters

Domain is the domain whose servers you want to list .

Returns

A space-separated list of the load sharing servers in the domain.

Example

```
servers = GetLoadsharingServer(dean@companyA.dom);  
Print(servers);
```

Result

server1 server2 mainserver

See also

GetLoadsharingServer

GetLocalDomain

GetLocalDomain returns the domain name associated with the IP address.

```
str = GetLocalDomain(IP_address)
```

Parameters

The single parameter is the IP address.

Returns

The name of the local domain which matches the the IP address, or an empty string if no domain does match.

Example

```
dom = GetLocalDomain(192.54.112.48);  
Print(dom);
```

Result

CompanyA.dom

See also

GetIPAddress

GetLocalIps

GetLocalIps returns the IP addresses which are local to this network.

```
str = GetLocalIps()
```

Parameters

None.

Returns

A list of the IP addresses in wildcard format.

Example

```
Print(GetLocalIps());
```

Result

```
192.54.30.66 192.54.32.*
```

See also

GetHostedIps

GetLocalAddr

GetLocalAddr returns the IP address which the user's Web browser has connected to the server on.

```
addr = GetLocalAddr()
```

Parameters

None.

Returns

The IP address on which the user's Web browser connected to the server.

Example

```
Print(GetLocalAddr());
```

Result

192.54.30.44

See also

GetHostedIps, GetLocalIps

GetLogonUser

GetLogonUser returns the session's account name.

```
user = GetLogonUser()
```

Parameters

None.

Returns

One of the following:

- The session's account in the form "domain\user". This is not necessarily verified.
- An empty string if the session has been created but no user has logged on.

Example

```
Print(GetLogonUser());
```

Result

companyA.dom\fred

See also

GetConnectionVariables

GetMailboxName

GetMailboxName returns the account's default mailbox (folder) name.

x = GetMailboxName(account)

Parameters

Account is the account name.

Returns

The mailbox name minus its ".mbx" extension.

Example

```
Print(GetMailboxName("companyA.dom\\joe"));
```

Result

inbox

See also

FolderList

GetMaxThreads

GetMaxThreads returns the maximum number of threads allowed.

```
x = GetMaxThreads()
```

Parameters

None.

Returns

The maximum number of threads allowed for any service.

Example

```
Print(GetMaxthreads());
```

Result

GetMemberRecord

GetMemberRecord opens a member record. You can then change its variables and write it back to the database.

mhandle = GetMemberRecord(db_handle,member_name)

Parameters

Db_handle

The handle created using **OpenMemberDb**.

Member_name

The member to find, given in standard account format.

Returns

Mhandle, the handle of the member's record, with these attributes:

Variable	Meaning
Member	The address in account format.
Type	The "Type" bitmap; see the <i>GMS Reference Guide</i> .
Digest	The current effective digest setting, TRUE or FALSE.
digest_set	Digests set by this member (not list default) TRUE or FALSE.
digest_type	Digest format — "Off", "Text", "Index" or "MIME".
digest_type_set	Digest type set by this member (not list default) TRUE or FALSE
Show	Show originating address in posts from this user TRUE or FALSE
show_set	"Show" set by this member (not list default) TRUE or FALSE.
Suspended	Member suspended TRUE or FALSE.
Ack	Acknowledge posts from this user TRUE or FALSE.
AcceptCommand	Command awaiting user confirmation.
DateConfirmSent	If this list required members to confirm their membership, this will contain the date that the confirmation request was sent to the user.
DateJoined	Date that the person joined the list. If the join has not taken place yet, then the date that the request was received.
NumBadMsgs	Number of returned messages for member
ForDays	If defined, the number of days before the person will automatically be removed from the list.
DateLeft	Date the user was removed/left the list.
LeftReason	Reason why the user was removed/left the list.
HelpSentTime	Time last help sent to member.
MyPassword	Password user must use to modify their list settings.
SuspendDays	If defined, the number of days to suspend posting of messages to the member.

Variable	Meaning
Name	From join message header if not also a Required or Optional-Field.
Organization	From join message header if not also a Required or Optional-Field.
ModerationDate	Date for expiry of moderation of commands other than Join/Leave
AcceptVals	Command parameters associated with command being moderated.

If you specify a variable which does not exist for the member, an empty string is returned. Assigning this creates the variable for the member and gives it the value.

Remarks

If the member does not exist, this function creates a blank record for them. In this case, **SetMemberRecord** must be called to save any changes to the member's record, assuming that the database was opened in read/write mode.

Example

```
dbase = OpenMemberDb(companyA.dom\stafflist,FILE_WRITE);
if (dbase)
{
    mem = GetMemberRecord(dbase,companyA.dom\joe);
    SetMemberRecord(dbase,mem);
}
CloseMemberDb(dbase);
```

See also

CloseMemberDb, GetAllMembers, OpenMemberDb, ReadNextMemberRecord, SetMemberRecord

GetMXRecord

GetMXRecord performs an MX lookup on the hostname specified and returns a list of the MX hosts for this address in priority order.

```
str = GetMXRecord(hostname)
```

Parameters

Hostname is the name of the host to look up.

Returns

A space-separated list of hostnames and their priorities. Each priority is separated from its host name by a semi-colon (;).

Remarks

The priority order is highest priority (lowest number) first.

Example

```
Print (GetMXRecord("companyA.dom"));
```

Result

```
mail.companyA.dom;10 mail.insnet.net;20
```

See also

GetHostname

GetOs

GetOS obtains the operating system.

n = GetOs()

Parameters

None.

Returns

One of the following:

1. WinNT.
2. Win95.

Remarks

Further operating systems may soon be added to the list.

Example

On a Windows NT server this will return "1":

```
GetOS();
```

GetOsStr

GetOsStr obtains a string identifying the operating system.

n = GetOsStr()

Parameters

None.

Returns

One of the following:

'winnt' includes NT3.51, NT4 and Windows 2000

'win95' includes win98

'solaris' includes sparc and intel versions

'unknown'

Remarks

Further operating systems may soon be added to the list.

Example

On a Windows NT server this will return "winnt":

```
GetOsStr();
```

GetPostFixes

GetPostFixes obtains a list of the domain's postfixes.

n = GetPostFixes(domain)

Parameters

domain

The domain for which postfixes should be listed.

Returns

a space separated list of postfixes used in that domain.

Remarks

Further operating systems may soon be added to the list.

Example

```
x = GetPostFixes("CompanyA.dom");  
print(x);
```

Result

postfix1 postfix2 postfix3

See also

AddPostfix, RemovePostfix, AddDomain

GetProcessorStr

GetProcessorStr obtains a string identifying the server processor.

```
n = GetProcessorStr()
```

Parameters

None.

Returns

One of the following:

'intel'

'alpha'

'unknown'

Remarks

Further operating systems may soon be added to the list.

Example

On an Intel machine this will return "intel":

```
GetProcessorStr();
```

See also

GetOsStr

GetProtocolText

GetProtocolText returns the protocol string for the current connection.

n = GetProtocolText()

Parameters

None.

Returns

the protocol string for the current connection (HTTP, HTTPS, FTP)

Example

```
print(getprotocolText());
```

See also

GetConnectionVariables, GetProtocolType

GetProtocolType

GetProtocolType returns an integer representing the current protocol type.

```
n = GetProtocolType()
```

Parameters

None.

Returns

an integer representing the current protocol type.

Example

```
Print(GetProtocolType());
```

Result

3

See also

GetConnectionVariables, GetProtocolText

GetProxyCacheSize

GetProxyCacheSize obtains the current cache size.

```
n = GetProxyCacheSize()
```

Parameters

None.

Returns

The current proxy cache size in bytes.

Example

```
print(getproxycachesize());
```

See also

ProxyAgeCache

GetRand

GetRand gets a random number

n = GetRand([modulus])

Parameters

Modulus(optional)

the optional modulus of the random number.

Returns

a random number

Example

```
print(getrand());
```

GetRealLogonUser

GetRealLogonUser obtains the real account name of the currently logged on user.

```
n = GetrealLogonUser()
```

Parameters

None.

Returns

The real account name of the currently logged on user.

Example

```
AddSession("joe@CompanyA.dom", "joespassword");  
x = GetRealLogonUser();  
print(x);  
Delsession();
```

Result

companyA.dom\joe

GetRemoteConnectionAddr

GetRemoteConnectionAddr returns the IP address of a connected user's Web browser.

```
addr = GetRemoteConnectionAddr()
```

Parameters

None.

Returns

The IP address of the Web browser belonging to the user who has connected.

Example

```
Print(GetRemoteConnectionAddress());
```

Result

188.78.12.35

See also

GetIPAddress

GetSessionID

GetSessionID gives the ID for the current session.

```
str = GetSessionID()
```

Parameters

None.

Returns

The Id for the current session.

Example

```
print(GetSessionID());
```

Result

8e573f13eaa0422958d93aab91bbe2af

GetSessionVariables

GetSessionVariables lists variables for the session. It's only used for debugging.

str = GetSessionVariables()

Parameters

None.

Returns

A space-separated list of GLOBAL and SESSION variables for the session.

Example

```
GetSessionVariables();
```

Result

Variable Name	Value
action	New schedule
HTTP_Accept	image/gif,image/x-xbitmap,image/jpeg, image/pjpeg, */*
HTTP_Accept_Charset	iso-8859-1,*,utf-8
HTTP_Accept_Language	en
HTTP_Connection	Keep-Alive
HTTP_Content_length	21
HTTP_Content_type	application/x-www-form-urlencoded
HTTP_Host	127.0.0.1:8000
HTTP_REFERER	http://127.0.0.1:8000/Dialup/Rule.mml? Change Open=Dialup&ChangeTabIndex=1
HTTP_User_Agent	Mozilla/4.01 [en] (WinNT; I)
LOCAL_ADDR	127.0.0.1

etc, etc.

See also

"Diagnostics" on page 302.

GetStatus

GetStatus returns the status of a service.

```
str = GetStatus(service)
```

Parameters

Service is one of these: ALL_SERVICE, SMTP_SERVICE, POST_SERVICE, POP_SERVICE, IMAP_SERVICE or WWW_SERVICE.

Returns

These four fields are shown after the service name:

- Service id.
- IP address.
- Status —IDLE, LOGN, HELO, etc. For a full list for each service, see the *Gordano Administrator's Guide*.
- Time — the time the thread has spent processing the transaction, in seconds.

Example

```
GetStatus(LIST_SERVICE);
```

Result

```
CU LIST 0 127.0.0.1 LOGN 25.32
```

See also

ServiceStatus

GetSupportInfo

GetSupportInfo gets a list of registered local support contacts.

```
str = GetSupportInfo()
```

Parameters

None.

Returns

a comma seperated list of addresses specified as contact addresses which Gordano support should reply to in response to support enquiries.

Example

```
print(GetSupportInfo());
```

Result

```
postmaster@CompanyA.dom,joe@CompanyA.dom,Jane@CompanyA.dom
```

GetUID

GetUID generates a unique string

GetUID()

Parameters

None

Returns

Unique string

Example

```
Print (GetUID());
```

Result

Unique string similar to:
14304950600013

GetUsersOfType

GetUsersOfType lists all accounts of the specified type in the domain.

```
str =  
GetUsersOfType(domain,typebits[,zerotype[,usertypes[,seperator[,bannedtypes]]]]);
```



Only a verified account can use this function.

Parameters

Domain

The domain name.

Typebits

A bitmask of account types. For example, a standard mail account with no aliases would take the value 262146, the decimal equivalent of bit 1 plus bit 18. See the *GMS Reference Guide* for full details.

Zerotype

If true, includes accounts with an account type of 0. These are usually deleted accounts. The mask cannot include these.

UserTypes

If true returns the user type in the results. Default is false.

Seperator

The seperator used to delimit the results.

BannedTypes

Bitmaps of types that will be banned from the results. Default value will vary due to the location of the script. If the script is located in a list sub directory default will ban all list and group types.

Set value to 0 to allow all types.

Returns

A colon-separated list of accounts.

Remarks

The function takes a list of users, a user attribute to check, the value of that attribute you are looking for and finally a separator. The output includes accounts from NT SAM and any authorised DLLs.

This function can be slow for large membership databases.

Examples

The following piece of MML would run through the entries in your useraccesslist checking the AccessRights attribute for any that have rights set to access the complete system and return a list of these using : as a separator.

```
FilterUsersOfType("\\UserAccessList", "AccessRights", ACCESS_RIGHT_SYSTEM, ":")
```

This uses a mask of 16 to find superlist accounts in the domain CompanyA.dom, and also has the zerotype set to find deleted users:

```
GetUsersOfType("companyA.dom", "16", TRUE, FALSE, :, 0);
```

Result

Listmaster and listsuper are superlists, Fred's is a deleted account:

```
listmaster,16:listsuper,16:fred,16
```

See also

FilterUsersOfType, GetAllMembers

IMIsAvailable

IMIsAvailable validates if a user is logged on to Instant Messaging and is available to receive messages.

IMIsAvailable(Email Address,[Timeout])

Parameters

Email Address

An address in the format user@domainname

Timeout (Optional)

Timeout in seconds.

Infinite if not selected

Returns

TRUE if the user is online

FALSE if the user is not online

Example

This checks if the user is online and available for 10 seconds

```
if (IMIsAvailable("postmaster@companyA.com",10))
{
    Print ("Online");
}
```

See also

IMSendMessage

ImportFolder

ImportFolder imports a mailbox from Eudora, Outlook Express or Netscape format.

Importfolder(*export*,*import*,*importtag*,*type*)

Parameters

export

the name of the folder messages are to be imported from.

import

the folder to which messages in the export folder are to be copied.

importtag

This tag is applied to all messages imported

type

The type of client from which you are importing the messages

0 = Eudora (v3)

1 = Netscape (v4)

2 = Outlook Express (v5)

3 = UNIX mailbox

Remarks

Since there is no absolute path option in this function you will need to copy the file to be imported under the user's directory. The suggested location is domain\username\Attachments\folderimport.

The imported file is deleted so make sure you keep a copy elsewhere if you want to retain it.

Returns

The number of messages that have been imported.

Example

```
report =  
importfolder("companyA.dom\joe\Attachments\folderimport\Inbox.dbx", "companyA.d  
om\joe\InBox", "", "2");  
print(report, " Outlook Express messages imported");
```

Result

16 Outlook Express messages imported

ImportMembers

ImportMembers imports a list of members into a list.

Importmembers(listname,memberlist,replace)

Parameters

listname

the name of the list the members are to be added to.

memberlist

the list of members for import in the format

user@domain[, Digest=x, Show=x, Ack=x, MyPassword=x,
Digest_Type=x, Suspended=x]

replace

if set to true existing member records matching any from the imported file will be replaced with the new information in the imported file.

Returns

false on error.

Remarks

Each member record needs to be on a line of its own

Example

```
import_addlist = "import@companyA.dom \r\n import1@companyA.dom";  
import_addlist = Trim(import_addlist,"r\n");  
import_addlist = Trim(import_addlist);  
importmembers("companyA.dom\\yourlist",import_addlist,1);
```

Results

Two members get added to the list, none of the optional switches such as Digest are set so the list defaults will apply.

IMSendMessage

IMSendMessage sends an instant message to a user.

IMSendMessage(From,To,Message)

Parameters

From

Email address the message is from.

To

Email address the message is to.

Message

Message content. The message is limited to 256 characters, any data over 256 characters will be truncated.

Returns

Nothing

Remarks

If the user is not online the messages will be deleted.

Example

```
send = IMSendMessage("postmaster@companyA.com","user1@CompanyA.com","Hello World");
```

Result

Message sent to user1@companyA.com showing "Hello World".

See also

IMIsAvailable

IncrementDate

IncrementDate increments one or more of the segments of a date and returns a valid date structure.

IncrementDate(date, day [,month] [,year])

Parameters

date

The date structure to be adjusted

day

A whole number of days to be added

month

A whole number of months to be added (optional)

year

A whole number of years to be added (optional)

Returns

An adjusted date structure

Example

```
d1 = Date(1964, 12, 28);  
d2 = IncrementDate(d1, 0, 3);  
  
print(d1);  
print(d2);
```

Result

```
1964-12-28  
1965-03-28
```

See also

AddDate

InStr

InStr searches for a string within a larger string and returns the location that the string starts in.

n = InStr(search_string,look_for_string[,flag])

Parameters

Search_string

The string to search.

Look_for_string

The string to look for.

Flag

If this is TRUE search in reverse order.

Returns

- 0 — if the string is not found.
- n — the character position at which <Look_for_string> starts.

Example

```
Print(InStr("hello world", "world"),<BR>);  
Print(InStr("banana bunch", "world"),<BR>);  
Print(InStr("hello world hello world", "world", TRUE),<BR>);
```

Result

```
7  
0  
19
```

See also

Match, Mid

Interpret

Interpret runs a specified script.

Interpret(script)

Parameters

Script

the script to be run

Returns

The output from the script

Example

```
manual = interpret(mid(H1,3), TRUE);
```

IsAbsoluteFilename

IsAbsoluteFilename tests whether a filename is absolute.

tf = IsAbsoluteFilename(filename)

Parameters

The name of the file to test.

Returns

TRUE if the pathname is absolute, otherwise FALSE.

Example

```
if (IsAbsoluteFilename(filepath))  
{  
    Print("Do not use absolute filenames");  
}
```

See also

The notes on absolute filenames at the start of this chapter.

IsConnected

IsConnected reports the status of a RAS (Remote Access Service) connection.

tf = IsConnected()

Parameters

None.

Returns

TRUE if there's an active RAS connection to the Internet.

Example

```
if (!IsConnected())
{
    if (Autoconnect("Demon internet", "Custom dialup"))
    {
        Print ("Connected");
    }
}
```

See also

AutoConnect, AutoDisconnect

IsDate

IsDate tests whether the variable specified is in valid date format.

tf = IsDate(date)

Parameters

Date is a string.

Returns

TRUE if the variable specified is in valid date format, otherwise FALSE.

Remarks

If you want to test for a valid date, for example to rule out dates like February 31st, use **IsValidDate** instead.

Example

```
Print (IsDate(" 1997-06-04 "),"<BR>");  
Print (IsDate("hello"),"<BR>");
```

Result

```
1  
0
```

See also

DateTimeFormat, IsValidDate

IsDialupEnabled

IsDialupEnabled tests whether dialup is enabled on the system.

```
tf = IsDialupEnabled()
```

Parameters

None.

Returns

TRUE if dialup is enabled, otherwise FALSE.

Example

```
if (!IsDialupEnabled())  
{  
    Print("Dialup is not operating");  
}
```

See also

AutoConnect, AutoDisconnect, IsConnected

IsDomain

IsDomain tests whether the domain specified is local.

IsDomain(domain)

Parameters

domain

the domain to check.

Returns

true if the domain is local else false

Example

```
if (isdomain("CompanyA.com"))
{
    print("Local");
}
else
{
    print("Non local");
}
```

IsInteger

IsInteger tests whether the variable specified is a valid integer number.

tf = IsInteger(variable)

Parameters

Variable is the number to be tested.

Returns

TRUE if the number is a valid integer, otherwise FALSE.

Example

```
Print IsInteger("78") CRLF;  
Print IsInteger(89) CRLF;  
Print IsInteger("hello") CRLF;
```

Result

```
1  
1  
0
```

IsIPAddress

IsIPAddress tests whether a variable is an IP address.

tf = IsIPAddress(**var**)

Parameters

Var is the variable to test.

Returns

TRUE if the IP address is valid, otherwise FALSE.

Example

```
y = IsIPAddress(192.44.55.12);  
yy = IsIPAddress("hello");  
Print("y", "<BR>", "yy");
```

Result

```
1  
0
```

See also

GetHostname

IsLoggedIn

IsLoggedIn tests whether a user is logged on.

```
tf = IsLoggedIn()
```

Parameters

None.

Returns

TRUE if a user is logged on and verified, otherwise FALSE

Example

```
AddSession("joe@CompanyA.dom", "joespassword");  
x = isloggedon();  
print(x);  
delsession();
```

Result

1

IsMemberOfList

IsMemberOfList tests whether a named account is a member of a GLList or GLCommunicator list.

tf = IsMemberOfList(listname,listaddress)

Parameters

Listname

is the list account to check. membership of.

ListAddress

is the address of the person that you want to check is a member or not.

Returns

TRUE if the named account is a member of an NTList list on the server, otherwise FALSE.

Example

```
listName = "domain.dom\\list"
listAddress = "domain.dom\\member"

if (!IsMemberOfList(listName, listAddress))
{
    Print(listAddress, " is not a member of ", listName);
}
else
{
    Print(listAddress, " is a member of ", listName);
}
```

See also

GetAllMembers

IsValidDate

IsValidDate checks whether the supplied date is valid. A date of 31st February, for example, will be rejected.

tf = IsValidDate(string)

Parameters

String is a date string.

Returns

TRUE if the date is acceptable, otherwise FALSE.

Remarks

To test whether a date format is correct use **IsDate** instead.

Example

```
Print(IsValidDate("31-02-99"),<BR>);  
Print(IsValidDate("21-02-99"));
```

Result

0
1

See also

IsDate

IsValidEmailAddress

IsValidEmailAddress checks whether the supplied e-mail address contains the characters of a valid e-mail address.

```
tf = IsValidEmailAddress(string)
```

Parameters

String is a string.

Returns

TRUE if the e-mail address is acceptable, otherwise FALSE.

Remarks

The fact that an address is valid does not mean that the named account actually exists in the domain given.

Example

```
Print(IsValidEmailAddress("joexcompanyA.dom",<BR>));  
Print(IsValidEmailAddress("joe@companyA.dom"));
```

Result

0
1

See also

VerifyUser

IsValidPassword

IsValidPassword checks that a password matches the current system password policy.

```
tf = IsValidPassword(password)
```

Parameters

Password is a string containing the password.

Returns

TRUE if the password is acceptable, otherwise FALSE.

Remarks

A password policy can dictate, for example, that passwords:

- Have at least six characters.
- Contain at least one number.

Example

```
if (!IsValidPassword(pwd))  
{  
    Print("Password is invalid!");  
}
```

See also

EncryptPassword

IsValidStr

IsValidStr tests whether all the characters in a string are from a permitted list.

```
tf = IsValidStr(string,valid_chars)
```

Parameters

String

The string for validation.

Valid_chars

The string containing characters that are valid in a string.

Returns

TRUE if all the string's characters are from "valid_chars", otherwise FALSE.

Example

```
Print(IsValidStr("abc","abcde"));  
Print(IsValidStr("abz","abcde"));
```

Result

```
1  
0
```

See also

InStr, Trim

IsValidUserName

IsValidUserName tests whether the specified username contains invalid characters for a username.

IsValidUserName(name)

Parameters

name

The username to check for validity.

Returns

TRUE if the username is valid

Example

```
if (IsValidUserName("joe"))
{
    print("UserName OK");
}
else
{
    print("UserName contains invalid characters");
}
```

IsWildcard

IsWildcard tests whether a string contains a wildcard.

tf = IsWildCard(string)

Parameters

String is the string to search.

Returns

TRUE if a wildcard is found in the string, otherwise FALSE.

Remarks

Currently only the wildcards "*" and "?" are checked for.

Example

```
Print(IsWildcard(" abc* "),<BR>);  
Print(IsWildcard(" abz "));
```

Result

```
1  
0
```

See also

IsValidStr, WildcardMatch

KillScript

KillScript kills the specified script.

KillScript(ThreadId)

Parameters

ThreadId

The thread Id of the script to be killed.

Returns

Nothing

Remarks

Currently only the wildcards "*" and "?" are checked for.

Example

```
KillScript("184")
```

See also

ListRunningScripts, SetScriptPriority

LanguageName

LanguageName gives the full name for the language code specified.

LanguageName(shortname)

Parameters

shortname

the code for the language that a full name is required for.

Returns

full name of the language

Example

```
Print(LanguageName("en-us"));
```

Result

English(United States)

See also

SetSessionLanguage, ConfiguredLanguages, DefaultLanguage, NLS,

Left

Left returns a string containing n characters from the left hand side of the string.

str = Left(string,n)

Parameters

String

The string for dissection.

n

The number of characters required.

Returns

Str is the string n characters long.

Remarks

If the string is less than n characters long, the function returns the whole string.

Example

```
z = "World Wide Web";  
Print(left(z, 5));
```

Result

World

See also

Len, Right, Mid, Trim

Len

Len returns the number of characters in the supplied string.

n = Len(string)

Parameters

String is the string whose length is required.

Returns

The length of the string.

Example

```
Print(len("World Wide Web"));
```

Result

14

See also

Left, Right, Mid, Trim

ListRunningScripts

ListRunningScripts Lists any scripts that are currently running.

n = listRunningscripts(showself)

Parameters

Showself

if set to true the current connection is included.

Returns

A space separated list of the scripts that are running. The type of script, location the script was called from, loggedon user, date and threadID are given for each script returned. Each of these values is separated by a semi-colon.

Example

```
print(listrunningscripts(1));
```

Result

```
USER;\test.mml;companyA.dom\joe;2001-04-19+16%3a34%3a32;200
```

See also

KillScript, SetScriptPriority

ListVersion

ListVersion returns the major version number of NTList.

`n = ListVersion()`

Parameters

None.

Returns

A single digit, the major version number. For version 4.00.20, for example, this would be "4".

Example

```
Print(ListVersion());
```

Result

4

Location

Location terminates execution of the current script and abandons its contents, then loads the specified page into the Web browser.

Location(URL[,perm_or_temp])

Parameters

URL

The URL the Web browser goes to.

Perm_or_temp

The mode the browser uses, permanent or temporary. The default is temporary.

Returns

Nothing.

Example

```
Location("http://www.companyA.dom");
```

Result

The Web browser jumps to the given URL instead of displaying any of the script produced.

Log

Log writes a line to the script engine's log file. This function is useful for timed scripts.

Log(level,stringlist)

Parameters

Level

The log level — Progress, Statistics, Returns, Protocol, GMS Anti-Spam, DNS or Failures (see the *GMS Reference Guide* for full details). Specify this using the appropriate constant, for example LOG_PROGRESS.

Stringlist

A sequence of strings.

Returns

Nothing.

Remarks

If you specify PROGRESS_LOG and progress logging is turned on, the strings are written to the log file.

Example

```
Log(LOG_FAILURE,"Event failed",Time());
```


LoggedInUsers

LoggedInUsers lists all accounts currently logged onto the Web site.

```
str = LoggedOnUsers()
```

Parameters

None.

Returns

A space-separated list of accounts.

Remarks

You must be logged on to use this function.

Example

```
LoggedInUsers();
```

Result

Joe@companyA.dom Kate@companyA.dom Simon@companyX.net

See also

GetAllMembers

LSAppend

LSAppend adds the given string to the end of the space-separated list.

```
str = LSApend(string,string_to_append)
```

Parameters

String

A space-separated list of strings.

String_to_append

The string to be added.

Returns

A string containing the original string, a space, then the newly appended string.

Remarks

If the list is separated by anything other than spaces, use **LSAppend2**.

Example

```
LSApend("hello abc def", "opt");
```

Result

hello abc def opt

See also

LSAppend2, LSDelete, LSDeleteElement, LSElement, LSFind, LSFIRSTMATCH, LSLength, LSMATCH, LSOrder, LSReplace, LSSubset

LSAppend2

LSAppend2 adds the given string to the end of a list.

```
str = LSAppend2(string,string_to_append[,separator])
```

Parameters

String

A list of strings separated by spaces or some other character.

String_to_append

The string to be added.

Separator (optional)

The character which is used to separate elements in the string. If this is not specified, the separator is assumed to be a space.

Returns

A string containing: a string then <space> then the string_to_append.

Remarks

Unlike **LSAppend**, this function does not require the list to be separated by spaces.

Example

```
LSAppend2("hello;abc;def", "opt", ";");
```

Result

hello abc def opt

See also

LSAppend, LSDelete, LSDeleteElement, LSElement, LSFind, LSFirstMatch, LSLength, LSMatch, LSOrder, LSReplace, LSSubset

LSDelete

LSDelete removes an element from the list and returns the result.

```
str = LSDelete(string,item_to_remove[,separator])
```

Parameters

String

A list of strings separated by spaces or some other character.

Item_to_remove

The element to remove from the string.

Separator (optional)

The character which is used to separate elements in the string. If this is not specified, the separator is assumed to be a space.

Returns

String with specified element(s) removed.

Remarks

If the element is not found, no action is taken. If the <item_to_remove> appears more than once, each occurrence is removed.



When searching the list of strings for deletion, the string comparisons are not case-sensitive.

Example

```
LSDelete("hello abc def ABC ", "abc");
```

Result

hello def

See also

LSAppend, LSAppend2, LSDeleteElement, LSElement, LSFind, LSFIRSTMATCH, LSLength, LSMATCH, LSOrder, LSReplace, LSSubset

LSDeleteElement

LSDeleteElement removes the *n*th element from the list and returns the result.

str = LSDeleteElement(string,number[,separator])

Parameters

String

A list of strings separated by spaces or some other character.

Number

The position of the element to remove from the string.

Separator (optional)

The character which is used to separate elements in the string. If this is not specified, the separator is assumed to be a space.

Returns

String with specified element removed.

Remarks

If the element is not found, no action is taken.

Example

```
LSDelete("hello abc def", 2);
```

Result

hello def

See also

LSAppend, LSAppend2, LSDelete, LSElement, LSFind, LSFirstMatch, LSLength, LSMatch, LSOOrder, LSReplace, LSSubset

LSElement

LSElement obtains a specified element from a list.

str = LSElement(string,n[,separator])

Parameters

String

A list of strings separated by spaces or some other character.

N

The number of the element to return.

Separator (optional)

The character which is used to separate elements in the string. If this is not specified, the separator is assumed to be a space.

Returns

A string containing the element specified. If the element does not exist, this string will be empty.

Example

```
LSElement("the force is strong within you", 2);
```

Result

force

See also

LSAppend, LSAppend2, LSDelete, LSDeleteElement, LSFind, LSFIRSTMATCH, LSLength, LSMATCH, LSOrder, LSReplace, LSSubset

LSFind

LSFind returns the position of an element in a string.

n = LSFind(list_string,element[,separator])

Parameters

String

A list of strings separated by spaces or some other character.

Element

The string to look for.

Separator (optional)

The character which is used to separate elements in the string. If this is not specified, the separator is assumed to be a space.

Returns

- 0 — if the element is not found.
- N — the number of the element in the string. For example, a value of 1 shows that the first entry matched.

Remarks

If the element appears in the list more than once, only the first occurrence is returned.

Example

```
LSFind("hello abc def", "def");
```

Result

3

See also

LSAppend, LSApPEND2, LSDelete, LSDeleteElement, LSElement, LSFirstMatch, LSLength, LSMatch, LSOrder, LSReplace, LSSubset

LSFirstMatch

LSFirstMatch looks for the first part of a string that matches a wildcard string and returns the number of that element.

n = LSFirstMatch(str,match_str[,separator])

Parameters

Str

A list of strings separated by spaces or some other character.

Match_str

The wildcard string to look for.

Separator (optional)

The character which is used to separate elements in the string. If this is not specified, the separator is assumed to be a space.

Returns

The number of the first element that matches. Zero (0) means there's no match.

Example

```
LSFind("hello world", "*wo");
```

Result

2

See also

LSAppend, LSAppend2, LSDelete, LSDeleteElement, LSElement, LSFind, LSLength, LSMatch, LSOrder, LSReplace, LSSubset

LSLength

LSLength counts the number of elements in a list string.

n = LSLength(string[,separator])

Parameters

String

A list of strings separated by spaces or some other character.

Separator (optional)

The character which is used to separate elements in the string. If this is not specified, the separator is assumed to be a space.

Returns

The number of elements in a list string.

Example

```
LSLength("hello;abc;def", ";");
```

Result

3

See also

LSAppend, LSAppend2, LSDelete, LSDeleteElement, LSElement, LSFind, LSFirstMatch, LSMatch, LSOrder, LSReplace, LSSubset

LSMatch

LSMatch returns a string list containing only those elements that match the wildcard parameter given.

str = LSMatch(string,match[,separator])

Parameters

String

A list of strings separated by spaces or some other character.

Match

A wildcarded string.

Separator (optional)

The character which is used to separate elements in the string. If this is not specified, the separator is assumed to be a space.

Returns

A string listing the entries that matched.

Example

```
LSMatch("abc def aty", "a*");
```

Result

```
abc aty
```

See also

LSAppend, LSAppend2, LSDelete, LSDeleteElement, LSElement, LSFind, LSFirstMatch, LSLength, LSOrder, LSReplace, LSSubset

LSOrder

LSOrder returns all the elements in the string in ASCII order.

```
str = LSOrder(in[,separator])
```

Parameters

In

A list of strings separated by spaces or some other character.

Separator (optional)

The character which is used to separate elements in the string. If this is not specified, the separator is assumed to be a space.

Returns

The string of elements in ASCII order.

Remarks

The case used for elements is ignored.

Example

```
LSOrder("abc def aty");
```

Result

```
abc aty def
```

See also

LSAppend, LSAppend2, LSDelete, LSDeleteElement, LSElement, LSFind, LSFirstMatch, LSLength, LSMatch, LSReplace, LSSubset

LSPopElement

LSPopElement removes the first element of a list and assigns it to the given variable name.

```
str = LSPopElement(list,name[,separator])
```

Parameters

list

the string from which the first element should be removed.

name

the name of a variable to which the first list element will be assigned.

Separator (optional)

The character which is used to separate elements in the string. If this is not specified, the separator is assumed to be a space.

Returns

A variable with the name specified in "name" and a value of the first element in the list.

Example

```
mylist = "part1 part2 part3 part4";
while mylist != ""
{
    LSPopElement(mylist,myelement);
    print(myelement,"<br>");
}
```

Result

```
part1
part2
part3
part4
```

See also

LSPopElement, LSPopElement2, LSDelete, LSDeleteElement, LSElement, LSFind, LSFirstMatch, LSLength, LSMatch, LSReplace, LSSubset, LSPopElement

LSPushElement

LSPushElement adds the specified element to the first position in the list.

```
str = LSPushElement(list,element[,separator])
```

Parameters

list

the string to which the new element should be added.

element

the text to be added as the first element.

Separator (optional)

The character which is used to separate elements in the string. If this is not specified, the separator is assumed to be a space.

Returns

The list with the defined element in the first position.

Example

```
mylist="part2 ";  
x = LSPushElement(mylist,"part1 ");  
print(mylist);
```

Result

```
part1 part2
```

See also

LSAppend, LSAppend2, LSDelete, LSDeleteElement, LSElement, LSFind, LSFirstMatch, LSLength, LSMatch, LSReplace, LSSubset

LSReplace

LSReplace searches through a string and replaces any sequence that matches with a new string.

str = LSReplace(in,match,newstr)

Parameters

In

The string to search and replace elements in.

Match

The element to replace in the string.

Newstr

The string to replace elements which "match" with.

Returns

The resultant string.

Remarks

This can be used to change the separator character in a string.

Example

This example replaces the space separator with a semi-colon (;).

```
LSReplace("abc def ghi", " ", ";");
```

Result

```
abc;def;ghi
```

See also

LSAppend, LSApPEND2, LSDelete, LSDeleteElement, LSElement, LSFind, LSFirstMatch, LSLength, LSMatch, LSOrder, LSSubset

LSSubset

LSSubset obtains a given number of elements from a list.

```
str = LSSubset(str,from_elem,num_elems[,separator])
```

Parameters

Str

A list of strings separated by spaces or some other character.

From_elem

The starting element string.

Num_elems

The number of elements to return, including <from_elem>.

Separator (optional)

The character which is used to separate elements in the string. If this is not specified, the separator is assumed to be a space.

Returns

A substring containing <num_elems> elements from <str> starting with the <from_elem> element specified. If <from_elem> is not in the list, the function returns an empty value.

Example

```
LSSubset("abc def ghi jkl mno", "def", 2);
```

Result

```
def ghi
```

See also

LSAppend, LSAppend2, LSDelete, LSDeleteElement, LSElement, LSFIND, LSFirstMatch, LSLength, LSMatch, LSOrder, LSReplace

Match

Match compares two strings and returns a value depending on their relationship.

```
n = Match(string_1, string_2)
```

Parameters

String_1

The first string.

String_2

The second string.

Returns

- 1 — if string 1 > string 2.
- 0 — if string1 = string2.
- -1 — if string 1 < string 2.

Example

```
Print("0 = ", Match("Joe", "JOE" ), "<br>");  
Print("1 = ", Match("joe", "o" ), "<br>");  
Print("-1 = ", Match("joe", "joed"), "<br>");  
Print("0 = ", Match(" ", " "), "<br>");  
Print("-1 = ", Match(" ", "hello" ), "<br>");  
Print("1 = ", Match("hello", " "), "<br>");
```

Result

```
0 = 0  
1 = 1  
-1 = -1  
0 = 0  
-1 = -1  
1 = 1
```

See also

WildcardMatch

Md5Str

Md5Str returns the MD5 digest of the specified string.

n = Md5Str(string)

Parameters

String

The string for conversion.

Returns

the md5 digest of the specified string

Example

```
x = Md5Str("This is a string");  
print(x);
```

Result

41fb5b5ae4d57c5ee528adb00e5e8e74

See also

Filemd5

MemberFormat

MemberFormat extracts variables from a member record.

str = MemberFormat(string,member_handle)

Parameters

String

Defines information required on each member, using a " %" to indicate the field to be completed. The case-sensitive fields are:

Field	Meaning
%a	Acknowledge
%A	AcceptCommand
%b	NumBadMessages
%c	DateConfirmSent
%d	Domain
%f	ForDays
%g	DigestType
%h	HelpSentTime
%i	DigestSet
%l	DigestTypeSet
%J	DateJoined
%j	TimeJoined
%L	DateLeft
%l	TimeLeft
%m	DateToCommandModerator
%n	Name (userid)
%p	MyPassword
%r	LeftReason
%s	Show
%S	ShowSet
%t	Type
%T	String version of Type
%u	Suspended
%U	UDIL
%X	DateExpiry
%x	TimeExpiry

Member_handle

The handle for the member, opened using **GetMemberRecord**.

Returns

The string for the member in the specified format.

Example

```
mem = GetMemberRecord(dbase, "joe@companyA.dom");  
Print("Member joined on ", MemberFormat("%J", mem), "<BR>");
```

Result

Member joined on 1999-06-16

See also

GetMemberRecord, ReadNextMemberRecord, SetMemberRecord

Mid

Mid returns a specified number of characters starting from position <from> in the given string.

```
str = Mid(string,from[,how_many])
```

Parameters

String

The string to parse.

From

A number indicating the position to start copying to the new string. The first character is number 1.

How_many (optional)

The number of characters to copy to the new string. If this is not specified, copy the whole string to the result.

Returns

One or more characters from the string.

Example

```
Print(Mid("hello world", 1, 5), "<br>");  
Print(Mid("hello world", 7, 5), "<br>");
```

Result

```
hello  
world
```

See also

Left, Right, Len, Trim

MsgAddAttachment

MsgAddAttachment adds an attachment to a message.

MsgAddAttachment(handle,account,filename[,subdir])

Parameters

Handle

The handle produced using **MsgCreate**.

account

The account that owns the file.

filename

the filename of the file to be attached.

subdir(optional)

the subdirectory the file is in

Returns

Nothing.

Remarks

Function fails if the handle parameter is not a message or the message doesn't exist.

Example

```
msg=msgCreate("j@jtest.dom","j@jtest.dom","test add attach");
if(msg)
{
    MsgAddBody(msg, "FirstLine");
    msgaddattachment(msg,"CompanyA.dom\joe","file.txt");
    msgclose(msg,MSG_SEND);
}
```

Result

This produces a message with this body:

FirstLine

and the attachment of file.txt

See also

MsgCreate, MsgAddHeader, MsgAddRecipient, MsgCopy

MsgAddBody

MsgAddBody appends a line to the end of the message.

MsgAddBody(handle,line[,wrap])

Parameters

Handle

The handle produced using **MsgCreate**.

Line

The string to append to the message.

Wrap

Whether to wrap the text around after 70 characters. The default is TRUE.

Returns

Nothing.

Remarks

Folder messages are read-only. To write to one of these, make a copy of it using **MsgCopy** then modify that.

Example

```
Msg = MsgCreate("simon@simon.dom", "fred@wibble.dom", "Hello");
if (Msg)
{
    MsgAddBody(msg, "Hello Fred");
    MsgAddBody(msg, "How are you?");
    MsgAddBody(msg, "Simon");

    Print(Msg\Recipient, "<br>");
    Print(Msg\Date, "<br>");

    MsgClose(Msg,MSG_SEND);
}
```

Result

This produces a message with this body:

```
Hello Fred
How are you?
Simon
```

See also

MsgCreate, MsgAddHeader, MsgAddRecipient, MsgCopy

MsgAddFile

MsgAddFile attaches a file to a message.

MsgAddFile(handle,filename,type)



Only a verified account can add files with absolute path names.

Parameters

Handle

The handle produced using **MsgCreate**.

Filename

The name of the file to attach.

Type

FILE_TEXT or FILE_BINARY. The option FILE_TEXT uses MIME if required.

Example

This appends the file Myzip to the message:

```
msg = MsgCreate(to,from,"here's the file");
if (msg)
{
    MsgAddFile(msg,myzip,FILE_BINARY);
    MsgClose(msg);
}
```

See also

MsgCreate, MsgAddHeader, MsgAddBody, MsgAddRecipient

MsgAddHeader

MsgAddHeader adds a new clause to the header of the message.

MsgAddHeader(handle,clause[,line[,replace[,append]]])

Parameters

Handle

The message's handle, created using **MsgCreate**.

Clause

The clause to add. If there's already a clause with this name, it will be replaced by the new one.

Line

If line is NULL or empty, the function deletes the clause from the header.

Replace (True or False)

If set to TRUE adds a new header clause, if the clause already exists it will be replaced. If set to FALSE and the clause exists it will not be changed.

Append (True or False)

If set to TRUE the header will be appended to the existing header list, i.e. placed at the end of the list. If set to FALSE it will become the topmost header.

Note that if the 5th parameter is used that the behaviour of the 4th parameter changes from "replace" to "remove", i.e. it will remove the existing header.

Remarks

The MsgCompose function is required to commit the changes to the email

Returns

TRUE if the clause is added successfully, otherwise FALSE.

Example

This adds a subject clause:

```
Print("Inserting \"[MAIL-LIST]\" into subject");  
MsgAddHeader(Msg,"Subject","[MAIL-LIST] " + Msg\Subject);  
MsgCompose(Msg);
```

See also

MsgCreate, MsgAddBody, MsgAddRecipient

MsgAddRecipient

MsgAddRecipient adds a recipient e-mail address (RCPT clause) to a mail message.

MsgAddRecipient(handle,email_address)

Parameters

Handle

The message's handle, created using **MsgCreate**.

Email_address

The string containing the e-mail address to add to the delivery address, in the form "user@domain.dom".

Returns

TRUE if the address is added successfully, otherwise FALSE.

Example

```
Msg = MsgCreate("Sales <sales@companyA.dom>", "joe@  
companyA.dom", "Test message");  
MsgAddBody(Msg, "First line of body");  
for (i=0; i<199; i = i+1)  
{  
    MsgAddRecipient(Msg, i + "@companyA.dom");  
}  
Print("Spamming ", MsgClose(Msg,MSG_SEND), "e-mail addresses");
```

Result

Spamming 200 e-mail addresses

See also

MsgCreate, MsgAddHeader, MsgAddBody

MsgClose

MsgClose completes the creation of a message.

```
n = MsgClose(handle,action[,errorval[,delivery_domain]])
```

Parameters

Handle

The message handle created using **MsgCreate**.

Action

One of the following:

- MSG_SEND — put the message in the queue and ask Post to send it immediately. The message will be sent to all the recipients (using the same mechanism as the list server).
- MSG_QUEUE — just put the message in the queue. If many messages are being generated, follow this function with a call to **SendNotification** to tell Post to start either all queues or this specific queue.
- MSG_DROP — simply drop the message.



MSG_QUEUE and MSG_DROP can only be used with message handles created by either MsgCreate or MsgCopy. They are not suitable for use with the "email" object in an SMTP script.

ErrorVal (optional)

Variable used to hold the error returned if the message fails.

Delivery_domain (optional)

If you want to deliver the message directly, rather than going back through GMS, set this to TRUE. If you do this, the mail will not be checked by GMS Anti-Spam, Virus Scanner, etc.

Returns

The total number of messages generated. A value of 0 means a failure.

Remarks

Once the message has been processed, all its resources are released.

Example

This example creates a one line message and sends it to 200 people:

```
Msg = MsgCreate("Sales <sales@companyA.dom>", "joe@companyA.dom", "Test message");
```

```
MsgAddBody(Msg, "First line of body");  
for (i=0; i<199; i = i+1)  
{  
    MsgAddRecipient(Msg, i + "@companyA.dom");  
}  
Print("Spamming ", MsgClose(Msg,MSG_SEND), "e-mail addresses");
```

Result

Spamming 200 e-mail addresses

See also

MsgCreate, MsgAddHeader, MsgAddBody, MsgAddRecipient

MsgCompose

MsgCompose composes a partially constructed message into its final form.

msgcompose(handle)

Parameters

Handle

is the handle of the message to be composed.

Returns

Nothing.

Example

```
msg = msgCreate("joe@CompanyA.dom", "joe@CompanyA.dom", "test size");
if (msg)
{
    MsgAddBody(msg, "FirstLine");
    MsgCompose(msg);
    print(msgsize(msg));
    msgclose(msg, MSG_SEND);
}
```

See also

MsgCreate, MsgAddHeader, MsgAddBody, MsgAddRecipient, Msgsize

MsgCopy

MsgCopy copies an existing message to make a new message.

```
msg = MsgCopy(handle)
```

Parameters

Handle is the handle of the message to be copied.

Returns

A new message handle.

Remarks

While **MsgCreate** creates a brand new message, **MsgCopy** copies an existing message. This is most useful in script.mml from GMS Anti-Spam. When a message arrives this MML is run from SMTP to perform GMS Anti-Spam's filtering. In this case you are given a special variable setup called "email". This is a read-only value so if you wish to modify the message or send it to someone else you must copy it. This is what **MsgCopy** is for.

The message this creates must be freed by using a separate call of **MsgClose** to that used for the original message.

Example

In a GMS Anti-Spam script the "email" message object can be used to check the message headers and body for matching criteria. This example uses this to copy the message to the Postmaster:

```
if (email\recipient == "joe@companyA.dom")
{
    newMsg = MsgCopy(email);

    if (newMsg)
    {
        newMsg\rcpt = "postmaster@companyA.dom";
        MsgClose(newMsg, MSG_SEND);
    }
}
```

Result

See also

MsgCreate, MsgAddHeader, MsgAddBody, MsgAddRecipient

MsgCreate

MsgCreate starts generation of a new message.

Handle = MsgCreate(from,to,subject)

Parameters

From

The sender's e-mail address. This is used in the SMTP MAIL clause and the From: clause in the message header.

To

The destination of the e-mail address. This is used in the SMTP RCPT clause and the To: clause in the message header.

Subject

The subject of the message. This must not exceed 75 characters.

Return Values

If the message is created, the result value is a handle which the other **Msg** functions can use to access the message.

If the result is FALSE, no message was created and continuing to use the handle will cause a fatal script error.

Warnings

When the script completes execution, any message that has not been explicitly closed is removed. This may result in some loss of resources.

Remarks

You must include a call to **MsgClose** before the script completes.

A plain text message is created unless the message has attachments, in which case it will be MIME-encoded.

MsgCreate is one of several functions that can create a message object. Message handles can also be created by the Folder functions and are automatically passed to the server in scripts that handle messages, for example, filter scripts.

To add additional recipients use **MsgAddRecipient**.

The handle has a series of parameters that can be read or set as indicated in this table:

Parameter	Access	Description
lines	RO	Number of lines in the email message.
size	RO	Number of bytes in the message, including its headers.
recipients	RW	A space-separated, unordered list of recipients of the message in the standard account form.
mail	RO	Returns the email address that will be used for the MAIL protocol clause when the message is closed.
date	RO	Reads the date the message was composed and attempts to convert it from the many forms seen on the Internet into the standard MML format so that other MML functions can access the information.
subject	RO	The subject clause.
messagedate	RO	The date field in its original format.
messageid	RO	Returns the message's unique ID if one has been assigned.
udil	RO	Returns the unique ID used by the POP3 server to identify messages uniquely to POP3 clients.
status	RO	Returns information about whether the message has been read. If it has not been read, returns "UNREAD".
<n>	RO	Returns line 'n' in the message. Line 1 is the first line of the BODY of the message.
<clause>	RO	Returns a given clause from the header of the message (if it exists). For example, result\from would return the full "From:" clause of the message.

Example

This creates and sends a message to fred and updates some of the message parameters:

```
Msg = MsgCreate("simon@simon.dom", "fred@wibble.com", "Hello");
if (Msg)
{
    MsgAddBody(msg, "Hello Fred");
    MsgAddBody(msg, "");
    MsgAddBody(msg, "How are you");
    MsgAddBody(msg, "Simon");

    Print(Msg\Recipient, "<br>");
    Print(Msg\Date, "<br>");

    MsgClose(Msg, MSG_SEND);
}
```

See also

MsgCopy, MsgAddHeader, MsgAddBody, MsgAddRecipient

MsgEndOfLines

MsgEndOfLines checks whether the end of the message has been reached.

MsgEndOfLines(handle)

Parameters

Handle is the message handle.

Returns

TRUE if the end of the message has been reached, otherwise FALSE.

Example

```
line = MsgReadFirstLine(Msg);
while (!MsgEndOfLines(Msg))
{
    Print(ConvertToHTML(line));
    Print("\n");

    line = MsgReadNextLine (Msg);
}
```

See also

MsgReadFirstLine, MsgReadNextLine

MsgReadFirstLine

MsgReadFirstLine reads the first line of a message.

```
str = MsgReadFirstLine(handle)
```

Parameters

Handle is the message handle.

Returns

A string containing the line from the message.

Example

```
line = MsgReadFirstLine(Msg);
while (!MsgEndOfLines(Msg))
{
    Print(ConvertToHTML(line));
    Print("\n");

    line = MsgReadNextLine (Msg);
}
```

See also

MsgEndOfLines, MsgReadNextLine

MsgReadNextLine

MsgReadNextLine reads the next line of the message.

```
str = MsgReadNextLine(handle)
```

Parameters

Handle is the message handle.

Returns

A string containing the line from the message.

Example

```
line = MsgReadFirstLine(Msg);
while (!MsgEndOfLines(Msg))
{
    Print(ConvertToHTML(line));
    Print("\n");

    line = MsgReadNextLine (Msg);
}
```

See also

MsgEndOfLines, MsgReadFirstLine

MsgRemoveHeader

MsgRemoveHeader removes headers from a message.

MsgRemoveHeader (handle,clause)

Parameters

Handle

The message's handle

Clause

The clause to remove.

Returns

TRUE if the cluse is removed successfully, otherwise FALSE

Example

This removes the Received: message headers:

```
headerLeft = true;
while ( headerLeft )

{
    headerLeft = MsgRemoveHeader( email, "Received:" );
}

msgclose(email,MSG_SEND);
```

See also

MsgAddHeader.

MsgSetEncoding

MsgSetEncoding Sets the encoding for the specified message..

MsgSetEncoding(handle,encoding)

Parameters

Handle

is the message handle.

encoding

the type of encoding to use.

0 - None

1 - Automatic

2 - Split Lines

3 - Quoted Printable

Remarks

The encoding can only be set once.

Returns

Nothing.

Example

```
MsgSetEncoding(msg,"3")
```

See also

MsgCreate, MsgAddBody, MsgClose

MsgSize

MsgSize returns the size of the specified message.

Size = MsgSize(handle)

Parameters

Handle

is the message handle.

Returns

A string containing the size of the message.

Remarks

requires the use of MsgCompose before it is called.

Example

```
msg = msgCreate("joe@CompanyA.dom", "joe@CompanyA.dom", "test size");
if (msg)
{
    MsgAddBody(msg, "FirstLine");
    MsgCompose(msg);
    print(msgsize(msg));
    msgclose(msg, MSG_SEND);
}
```

Result

216

See also

Msgcreate, MsgAddBody, MsgCompose, MsgClose

Nls

Nls print a language specific message identified by "messageID"

```
nls(messageID[,param1[,param2[,...[paramn...]]]])
```

Parameters

MessageID

is the id of the message to be returned

param1, param2, Paramn(optional)

Replaceable parameters in the form %1%, %2% etc where
%1% corresponds to *param1*, %2% corresponds to
param2 etc

Returns

the language specific message identified by messageid.

Example

```
nls("2");
```

See also

DefaultLanguage, ConfiguredLanguages, LanguageName,
SetSessionLanguage

ODBCInstalled

ODBCInstalled tests whether ODBC is installed on the system.

```
tf = ODBCInstalled()
```

Parameters

None.

Returns

TRUE if ODBC is installed, otherwise FALSE.

Remarks

If ODBC is not installed, this function tries to load it.

Example

```
if (ODBCInstalled())  
{  
    Print("ODBC is installed");  
}
```

OpenMemberDB

OpenMemberDB opens the specified member database, whether this is a flat file or an SQL database.

handle = OpenMemberDB(account,access_type,repair)

Parameters

Account

A list name account, for example "test.com\test-list".

Access_type

FILE_READ (read only) or FILE_WRITE.

Repair

Alters the order of records. Possible values are:

DO_NOT_REPAIR - This is the option to choose if you are opening just to read the DB with ReadNextMemberDB (you are not calling WriteNextMemberDB) and the order of the records returned is not important.

REPAIR_ON_WRITE - This is the option to use if you are opening to use with GetNextMemberRecord or both ReadNextMemberRecord and WriteNextMemberRecord.

REPAIR_IMMEDIATELY - This is the option to choose if you are opening just to read the DB with ReadNextMemberDB (you are not calling WriteNextmemberDB) and the order of the records returned has to be in domain/account order.

Returns

- Handle — the handle for the database. This will be used in all other membership file functions (see the list below). The handle object sets handle\EOF TRUE if the end of the member file is reached.
- FALSE — if it fails to open the database, for example because the connection to the SQL server failed, or the Access database does not exist.

Remarks

This function locks the database until a **MsgClose** is issued on the message handle.

Example

This example opens a member database and prints all its member records:

```
mydb = OpenMemberDB("test.com\\testlist",FILE_READ);
```

```
if (mydb)
```



```
{
    while (mymember = ReadNextMemberRecord(mydb))
    {
        Print(" Account: ",mymember\member," <BR>");
    }

    CloseMemberDB(mydb);
}
```

See also

CloseMemberDb, GetAllMembers, GetMemberRecord,
SetMemberRecord, ReadNextMemberRecord

OpenZip

OpenZip opens the specified Zip file.

OpenZip(file)

Parameters

File is the handle of the Zip file.

Returns

Nothing.

Remarks

If the ZIP archive file does not already exist, this function creates it.

You cannot use MML to do things like extract files from a Zip file.

Example

```
zip = OpenZip("\\Test.zip");
if (zip)
{
    if (!AddZip(zip, "\\Timed.txt"))
    {
        Print("Couldn't add to zip");
    }
    CloseZip(zip);
    Print("Opened zip.");
}
else
{
    Print("Failed to open zip");
}
```

See also

AddZip, CloseZip

Print

Print finds the value of each parameter in turn and outputs the result to the Web server.

Print(x,y,z,...)

Parameters

There can be any number of parameters, separated by commas.

Returns

Nothing.

Example

```
a = 5;  
b = 4;  
Print("a + b = ", a+b);
```

Result

```
a + b = 9;
```

ProxyAgeCache

ProxyAgeCache deletes any proxy cache entry older than the given number of days.

n = ProxyAgeCache(days,how)

Parameters

Days

The number of days.

How

PROXY_PURGE_ALL, PROXY_PURGE_CREATED or PROXY_PURGE_LAST_USED.

For example, if you specify PROXY_PURGE_CREATED and 365 days, all entries created more than 365 days ago are deleted, even those which have been used more recently than this.

Returns

The cache size in bytes.

Example

This removes all entries which are older than 90 days:

```
ProxyAgeCache(90,PROXY_PURGE_ALL);
```

PurgeDNSCache

PurgeDNSCache clears the DNS cache.

PurgeDNSCache()

Parameters

None.

Returns

Nothing.

Example

```
PurgeDNSCache();
```

ReadNextMemberRecord

ReadNextMemberRecord reads the next member record in a list database.

```
str = ReadNextMemberRecord(db_handle)
```

Parameters

Db_handle is the handle of the database opened by **OpenMemberDb**.

Returns

- FALSE — if the end of the file is reached or there's an error.
- Member_handle — the handle of the member's record. This has the following attributes:

Variable	Meaning
Member	The address in account format.
Type	The "Type" bitmap; see the <i>GMS Reference Guide</i> .
Digest	The current effective digest setting, TRUE or FALSE.
digest_set	Digests set by this member (not list default) TRUE or FALSE.
digest_type	Digest format — "Off", "Text", "Index" or "MIME".
digest_type_set	Digest type set by this member (not list default) TRUE or FALSE
Show	Show originating address in posts from this user TRUE or FALSE
show_set	"Show" set by this member (not list default) TRUE or FALSE.
Suspended	Member suspended TRUE or FALSE.
Ack	Acknowledge posts from this user TRUE or FALSE.
AcceptCommand	Command awaiting user confirmation.
DateConfirmSent	If this list required members to confirm their membership, this will contain the date that the confirmation request was sent to the user.
DateJoined	Date that the person joined the list. If the join has not taken place yet, then the date that the request was received.
NumBadMsgs	Number of returned messages for member
ForDays	If defined, the number of days before the person will automatically be removed from the list.
DateLeft	Date the user was removed/left the list.
LeftReason	Reason why the user was removed/left the list.
HelpSentTime	Time last help sent to member.
MyPassword	Password user must use to modify their list settings.
SuspendDays	If defined, the number of days to suspend posting of messages to the member.

Variable	Meaning
Name	From join message header if not also a Required or Optional-Field.
Organization	From join message header if not also a Required or Optional-Field.
ModerationDate	Date for expiry of moderation of commands other than Join/Leave
AcceptVals	Command parameters associated with command being moderated.

Remarks

If the database has just been opened, the call returns its first record. All other variables are taken from the member record (which has an empty string for a variable not present).

Example

This example opens a member database and prints the member field of all its records:

```
mydb = OpenMemberDB("test.com\\testlist",FILE_READ);

if (mydb)
{
    while (mymember = ReadNextMemberRecord(mydb))
    {
        Print(" Account: ",mymember\member," <BR>");
    }

    CloseMemberDB(mydb);
}
```

See also

OpenMemberDb, CloseMemberDb, GetAllMembers, GetMemberRecord, SetMemberRecord

RegGetVal

RegGetVal returns the value of the setting within the GMS Registry hierarchy.

RegGetValue(key,type,name)

Parameters

key

the key relative to the Internet-Shopper registry key. e.g. MailLDAPAuth

type

1 for String values and 2 for DWORDS

name

name of value e.g. CacheExpiry

Returns

value

Example

```
RegGetVal(("mail\\" & key), regtype, regname)
```

See also

RegSetVal

RegSetVal

RegSetVal sets the value of a setting within the GMS Registry hierarchy.

RegSetValue(key,type,name,value)

Parameters

key

the key relative to the Internet-Shopper registry key. e.g. MailLDAPAuth

type

1 for String values and 2 for DWORDS

name

name of value e.g. CacheExpiry

value

value e.g. 2

Returns

nothing

Example

```
RegSetVal(("mail\\" & key), regtype, regname, regvalue)
```

See also

RegGetVal

RemovePostFix

RemovePostFix removes the specified postfix from the specified domain.

RemovePostFix(domain,postfix)

Parameters

domain

the domain from which the postfix is to be removed

postfix.

the postfix to be removed.

Returns

Nothing.

Example

```
removepostfix(" companyA.dom ", "postfix1 ")
```

See also

GetPostFixes, AddPostFix, DelDomain

Resolve

Resolve feeds the given string through the script server's "# resolving" algorithm.

```
str = Resolve(string[,preservecase])
```

Parameters

String

A string containing variable names that need to be resolved (that is, containing variable names inside "#" marks).

preservecase

If set to true the case of the string will be preserved else lowercase will be returned. If omitted lowercase is returned.

Result

A string containing the result of the resolution.

Remarks

This is useful for filename handling — see GMS Anti-Spam/filterdomain.mml.

Resolution changes "\\" to "\".

Example

The two Print statements here produce the same result, but the line using **Resolve** is neater:

```
username = "joe";  
domain = domain.dom;  
Print("#domain#", "\", "#username#");  
Print(Resolve("#domain#\\"#username#"));
```

Returns

```
domain.dom\joe  
domain.dom\joe
```

See also

IsValidStr

Right

Right returns characters from the right of the given string.

```
str = Right(string,n)
```

Parameters

String

The string to examine.

N

The number of characters to return.

Returns

The *n* rightmost characters of the given string

Remarks

If the string is shorter than *n* characters, the function returns the whole string.

Example

```
String = Right("Prepare",4);  
Print(String);
```

Result

pare

See also

Len, Left, Mid, Trim

RunExecutable

RunExecutable runs an executable file.

RunExecutable(filename[,errorcode,allowabsolutepaths])



Only a verified account can use this function.

Parameters

Filename - the name of the executable including the path.

Errorcode - defines a variable to hold the error code returned by the executable.

Allowabsolutepaths - if true allows paths outside of the Gordano directories to be used.

Returns

TRUE if successful, otherwise FALSE.

Remarks

Script execution stops until the executable has run and returned its return code. Paths can either be in the form of MML paths, i.e. \\bin\\loadfile.exe or absolute paths as in the example below. The System search paths are not used.

Example

```
test = RunExecutable(c:\temp\loadfile.exe,errorcode,TRUE);
```

See also

AddSession

SearchFile

SearchFile searches a file for the given string.

n = SearchFile(filename,string[,absolutePath])

Parameters

Filename

The file to search.

String

The string to look for. This can contain wildcards.

AbsolutePath (optional)

If set to TRUE this lets a verified account use absolute paths.

Returns

The number of the first line in the file which contains the string.

Example

```
Print("Date of birth found on line ",(SearchFile(Customer.txt,"Date of birth")," <br>");
```

Result

Date of birth found on line 28.

See also

InStr, IsValidStr

SendNotification

SendNotification sends a message to the specified service.

SendNotification(service,message)

Parameters

Service

The name of the service to send the message to. This is a bit field with the values POST_SERVICE, SMTP_SERVICE etc.

Message

The string to be sent to the service (for example, "QA companyA.dom" to start a queue).

Service	IPC	Description
ALL_SERVICE	DD domain	Delete domain 'domain'
ALL_SERVICE	DN domain	New domain 'domain'
ALL_SERVICE	DU domain	Update domain 'domain'
ALL_SERVICE	GD variable	Delete global variable 'variable'
ALL_SERVICE	GN variable	New global variable 'variable'
ALL_SERVICE	GU variable value	Update global variable 'variable'
All excl www	CU	Send current service status to www
All excl post/list	LS domain	Reload load-sharing file for domain 'domain'
SMTP_SERVICE	ES id	Save eSarah ID
SMTP_SERVICE	RF	Reload redirect file
SMTP_SERVICE	RB	Reload DNSBL servers file
SMTP_SERVICE	RW domain	Reload restricted word file for domain 'domain'
SMTP_SERVICE	AD user domain client_ip pop/imap	Inform SMTP about user dynamic IP
SMTP_SERVICE	FF domain	Reload footer file for domain 'domain'
POST_SERVICE	PS	Reload the postservers file

Service	IPC	Description
POST_SERVICE	QA queue	kick queue 'queue'
POST_SERVICE	QN queue	kick queue 'queue'
POST_SERVICE	QK	Kick all queues
POST_SERVICE	QS	From SMTP - kick queue specified by ETRN command
LIST_SERVICE	SMTP, folder, new, cmd, from, to, util, type	Add specified message to list queue.
LIST_SERVICE	ML	Re-read all queues.

Returns

TRUE if the message is sent successfully, otherwise FALSE.

Remarks

The message is service- and GMS-specific.

Example

This starts the domain CompanyA.dom's POST queue:

```
SendNotification(POST_SERVICE, "QA CompanyA.dom");
```


ServerDSNExists

ServerDSNExists checks whether Data Source Name (DSN) is enabled on the server.

tf = ServerDSNExists()

Parameters

None.

Returns

TRUE if DSN exists, otherwise FALSE.

Remarks

DSN is used with ODBC databases; see the *GMS Reference Guide* for details.

Example

```
if (ServerDSNExists(" "))
{
    Print("DSN is enabled.<br>");
}
```

See also

ODBCInstalled, ServerValidUser

ServerValidUser

ServerValidUser checks whether the DSN, username and password are valid.

tf = ServerValidUser(DataSourceName,UserID,Password)

Parameters

DataSourceName

The Data Source Name to be checked.

UserID

The UserID for the DSN to be checked.

Password.

The password for the DSN to be checked.

Returns

TRUE if DSN, UserID and Password are valid.

Remarks

DSN is used with ODBC databases; see the *GMS Reference Guide* for details.

Example

```
if (ServerValidUser("MyDSN","Joe","joesPassword"))
{
    Print("DSN, User and Password are valid.<br>");
}
```

See also

ODBCInstalled, ServerDSNExists

ServiceStart

ServiceStart starts a Gordano service.

ServiceStart(Service)



This does not apply to Windows 95 or Windows 98.

Only a verified account can use this function.

Parameters

Service is POST, SMTP, IMAP, POP or LIST or any other service. Specify it as a text name instead of, for example, SMTP_SERVICE.

Returns

TRUE — if the service starts successfully, or was already running.

FALSE — if it takes more than 60 seconds to start the service.

Remarks

If the service is already running the function returns immediately, returning TRUE.

Example

```
if (ServiceStart("SMTP"))
{
    Print("Started SMTP service.<br>");
}
```

Result

Started SMTP service.

See also

AddSession, ServiceStop

ServiceStatus

ServiceStatus determines whether a Gordano service is running.

tf = ServiceStatus(service)



This does not apply to Windows 95 or Windows 98.

Only a verified account can use this function.

Parameters

Service is POST, SMTP, IMAP, POP or LIST or any other service. Specify it as a text name instead of, for example, SMTP_SERVICE.

Returns

- TRUE — the service is running.
- FALSE — an error occurred or the service is not running.

Example

```
if (ServiceStatus("SMTP"))  
{  
    Print("SMTP service running.<br>");  
}
```

Result

SMTP service running

See also

AddSession, GetStatus, ServiceStart, ServiceStop

ServiceStop

ServiceStop stops the specified Gordano service.

ServiceStop(service)



This does not apply to Windows 95 or Windows 98.

Only a verified account can use this function.

Parameters

Service is POST, SMTP, IMAP, POP or LIST or any other service. Specify it as a text name instead of, for example, SMTP_SERVICE.

Returns

TRUE — if the service stops successfully.

FALSE — if it takes more than 60 seconds to stop the service.

Remarks

If the service is currently not running, the function returns straight away, returning TRUE.

Example

```
if (ServiceStop("SMTP"))
{
    Print("Stopped SMTP service.<br>");
}
```

Result

Stopped SMTP service.

See also

AddSession, ServiceStart

SetHTTPCacheable

SetHTTPCacheable sets the current connection cacheable flag.

```
SetHTTPCacheable(value);
```

Parameters

value

true or false.

Returns

Nothing.

Example

```
setHTTPCacheable(1)
```

Result

Makes the current connection cacheable.

SetHTTPCookie

SetHTTPCookie installs a cookie on the client machine containing useful information for retrieval at the next visit.

```
SetHTTPCookie(GordanoSearchString,data[,ExpiryDate,path,domain]  
);
```

Parameters

GordanoSearchString

The string that GetHTTPCookie will search for when retrieving the cookie.

data

Any information you want to store in the cookie.

ExpiryDate

The date at which the cookie expires. If this is passed then the cookie won't be retrieved.

path

Optional. The Path attribute specifies the subset of URLs to which this cookie applies. Set this to "/" to allow the cookie's use through the whole site.

domain

Optional. The Domain attribute specifies the domain for which the cookie is valid. An explicitly specified domain must always start with a dot.

Returns

Nothing.

Example

```
SetHTTPCookie("GordanoSignon",signoninfo,"Fri, 01-Jan-2010 00:00:00  
GMT", "/");
```

Result

Creates a cookie on the client's machine which can then be retrieved using GetHTTPCookie. Note that clients can be configured not to allow the setting of cookies.

See also

GetHTTPCookie

SetHTTPResponseStatus

SetHTTPResponseStatus forces the HTTP response code and message to the values specified.

```
SetHTTPResponseStatus(errorcode, "error description");
```

Parameters

errorcode

The HTTP error code appropriate to the current status

error description

Short text summary of the error encountered.

Returns

Nothing

Example

```
def PrintErrorPage(information)
{
    SetHTTPResponseStatus(404, "Not Found");
    print("<h1>404 Not Found</h1>\r\nThe server was unable to process the
request.<p>The error received was: ", information, "<p>\r\n");
    print("Please contact the system administrator");
}

if (pagecontents == " ")
{
    PrintErrorPage("no data");
    end;
}
```

See also

AddHTTPHeader

SetLogType

SetLogType sets the log type for the current connection.

```
SetLogType(logtype);
```

Parameters

logtype

defines which log file log entries for the www service should be written to

DEFAULT_LOG = WW log

WEBMAIL_LOG = WM log

DIALUP_LOG = DU log

Returns

Nothing

Example

```
SetLogType(WEBMAIL_LOG);
```

SetMemberRecord

SetMemberRecord sets information about a member.

SetMemberRecord(db_handle,member_handle)

Parameters

Db_handle

The handle created by **OpenMemberDb**.

Member_handle

The handle created by **GetMemberRecord** (not **ReadNextMemberRecord**).

Remarks

The database must be opened in read/write mode.

GetMemberRecord must be called before this function to generate a member handle.

Returns

Nothing.

Example

This changes a single member's record in the database:

```
mydb = OpenMemberDB("test.com\\testlist",FILE_READ);

if (mydb)
{
    mymember = GetMemberRecord(mydb,"xyz.com\\esleyf");

    if (mymember)
    {
        mymember\\eyecolour = "blue";
        SetMemberRecord(mydb,mymember);
    }

    CloseMemberDB(mydb);
}
```

See also

OpenMemberDb, CloseMemberDb, GetAllMembers,
GetMemberRecord, ReadMemberRecord

SetPassword

SetPassword changes an account's password.

SetPassword(account, domain, old_password, new_password)



Only a verified account can use this function.

Parameters

Account

The account's name.

Domain

The account's domain.

Old_password

The current password.

New_password

The password which is to replace the existing password.

Returns

TRUE if the password is changed successfully, otherwise FALSE. If the old password given is invalid, the function fails.

Remarks

You can change the passwords of SAM database users too.

Example

This changes Joe's password from "wh1sky" to "cat4dog":

```
SetPassword(joe, test.dom, wh1sky, cat4dog);
```

See also

AddSession, EncryptPassword

SetScriptPriority

SetScriptPriority changes the script thread priority to the specified value.

SetScriptPriority(value)

Parameters

value

is the script thread priority to be applied. Can take one of the following:

- script_priority_high
- script_priority_highest
- script_priority_normal
- script_priority_low
- script_priority_lowest

these map directly onto NT thread priorities. Thread priorities are reset to normal when a script ends. All script types start off as NORMAL.

Returns

Nothing

Example

```
SetScriptPriority(SCRIPT_PRIORITY_LOWEST);
```

See also

KillScript, ListRunningScripts

SetSessionLanguage

SetSessionLanguage Set the current session language to that specified.

SetSessionLanguage(shortname)

Parameters

shortname

the short name for the language to be used

Returns

Nothing

Example

```
SetSessionLanguage("en-us");
```

See also

LanguageName, ConfiguredLanguages, DefaultLanguage, NLS

Sleep

Sleep pauses GMS for a given length of time.

Sleep(millisecs)

Parameters

Millisecs is the time to pause for in milliseconds.

Returns

Nothing.

Remarks

This is not usually used in Web page generation, only in scripts that are timed (for example, a script which initialises an Internet connection at a specific time).

The function stops the script for the specified time, but does not release its resources etc.

Example

```
Sleep(24000);
```

SQLCreateDb

SQLCreateDb creates a SQL database.

SQLCreateDb(Createstmt,DSN,user,password,TableName,Columnlist)

Parameters

CreateStmt

The SQL statement to create a Database

DSN

The Data Source Name to use to connect to the database.

User

The username associated with the DSN

Password

The password associated with the DSN

TableName

The name of the table to create.

Columnlist

A list of columns to add to the table.

Returns

Nothing.

Example

```
table_created = SQLCreateDB("CREATE TABLE %s (%s int IDENTITY(1,1) PRIMARY KEY  
NONCLUSTERED, %s char(20) NOT NULL, %s char(20) NOT NULL, %s int NOT NULL)",  
"mydsn", "joe", "dsnpassword", "members", "UserID,UserName,Domain,Type");
```

```
if (!table_created)  
{  
    print("failed to create DB");  
}
```

Result

Table members created with four fields. Note the first %s in the *CreateStmt* is substituted with the *Tablename* when the function runs, The other %s entries relate to the fields entered in *columnlist*. i.e. the second %s relates to *UserId*, the third *UserName* etc. There is no limit to the number of fields that can be added but the *CreateStmt* has to be edited to tie up with the *Columnlist*. The example above might be used with a custom ODBC type list as in *GLCommunicator*.

SQLExec

SQLExec allows the execution of arbitrary SQL statements on a predefined ODBC connection.

SQLExec(Datasource,Username>Password,Statement,MaxRows,RowSep,ColSep,ColumnNames,Rows);

Parameters

Datasource

Data Source Name (ODBC)

Username

Database username, if required

Password

Database password, if required

Statement

SQL statement to run (Select, Insert, etc)

MaxRows

Maximum number of rows to return, only required for a select statement. To select all rows use -1.

RowSep

Character to use to separate rows in string (needs to be a safe character, ie pipe, colon, space, etc)

ColSep

Character to use to separate columns in string (needs to be a safe character, ie pipe, colon, space, etc)

ColumnNames

Variable to place column names into

Rows

Variable to place SQL results into.

Returns

Boolean True or False

Remarks

Example

```
<html>
<body>
<#
  def DoQuery(statement)
  {
```



```

datasource = "xxxxxx";
username = "xxxxxx";
password = "xxxxxx";
maxRows = -1;
rowSep = "|";
colSep = ":";

ok = SQLExec(datasource, username, password, statement, maxRows, rowSep,
colSep, columnNames, rows);

print("SQL statement is: ", statement, "<br>");

if (!ok)
{
    print("SQL Failed<br>");
}

length = LSLength(columnNames, colSep);

#>
<table border=1>
<#
    for (i = 1; i <= length; i = i + 1)
    {
        col = URLDecode(LSElement(columnNames, i, colSep));

        print("<th>", col, "</th>");
    }

    length = LSLength(rows, rowSep);

    for (i = 1; i <= length; i = i + 1)
    {
        print("<tr>");

        row = LSElement(rows, i, rowSep);
        rowLen = LSLength(row, colSep);

        for (j = 1; j <= rowLen; j = j + 1)
        {
            col = URLDecode(LSElement(row, j, colSep));

            if (col == " ")
            {
                col = "&nbsp;";
            }

            print("<td>", col, "</td>");
        }

        print("</tr>\n");
    }
#>
</table>
<br>
<br>
<#
}

DoQuery("show tables");

#>
</body>

```

Result

SQL statement is: show tables

Tables_in_GLWebMail
abbooks
abrawvcards
abuserids
abvcards
calaccessrights
calalarmgroupids
calalarms
calattendeers
calevents
calselectedrights
email
imcontacts

See also

SQLCreateDb

Time

Time returns the current time or outputs the time specified.

t = Time([hours, minutes, seconds])

Parameters

If no parameters are specified, the function returns the current time to the nearest second.

If the parameters are specified, the function sets the system time to that time. In this case the date portion will be set to 0000-00-00

Example

```
now = Time();  
mydate = Time(12,30,0 );  
Print(now, "<BR>");  
Print(mydate, "<BR>");
```

Result

```
1999-06-05  
0000-00-00 12:30:00
```

See also

Date, DateTime, DateTimeFormat

TlsEnabled

TlsEnabled returns true if the tls dll is available. Required to enable SSL (Secure Socket Layer) connections

n = tlsenabled()

Parameters

None

Returns

TRUE is the Transport Layer Security (tls) dll is available, otherwise false.

Example

```
x = tlsenabled()  
Print(x);
```

Result

1

ToInt

ToInt converts the specified variable to an integer.

```
n = ToInt(var)
```

Parameters

Var is the variable to convert. This is normally a string.

Returns

An integer.

Example

```
prep = "256";  
prep1 = "1024";  
n = ToInt("prep + prep1");  
Print(n);
```

Result

1280

See also

IsInteger

Trim

Trim removes any "white space" (see below) from either end of a string.

str = Trim(string[,chars])

Parameters

String

The string to remove characters from.

Chars

If present, this defines the characters that may be removed from the string. If it's not present, all "white spaces" are removed.

Returns

A string.

Remarks

"White space" includes spaces, tabs, carriage returns and line feeds.

Example

```
Print(Trim(" hello "), "<BR>");  
Print(Trim("aaahelloa", "a");
```

Result

```
hello  
hello
```

See also

Left, Right, Mid, Len

UrlDecode

UrlDecode decodes an encoded URL.

```
str = UrlDecode(string)
```

Parameters

String is the URL to decode.

Returns

The decoded string.

Remarks

This reverses the **UrlEncode** function.

This is not encoding/decoding as the term is used for passwords. It means making the string URL-compliant.

Example

```
URLstring = URLDecode("Hello+world");  
Print(URLstring);
```

Result

Hello world

See also

UrlEncode

UrlEncode

UrlEncode encodes a URL.

```
str = UrlEncode(string)
```

Parameters

String is the URL to decode.

Returns

This is not encoding/decoding as the term is used for passwords. It means making the string URL-compliant.

Remarks

This reverses the **UrlDecode** function.

Example

```
URLstring = URLEncode("Hello world");  
Print(URLstring);
```

Result

Hello+world

See also

UrlDecode

VerifyUser

VerifyUser checks that an account is valid.

VerifyUser(account,password)

Parameters

Account

The account name.

Password

The account's password.

Returns

TRUE if the account is valid and its password is correct, otherwise FALSE.

Remarks

If the account is "user", the function uses the connection's IP address for the domain name.

Example

```
if (!VerifyUser(dean@test.dom,wh1sky))
{
    Print ("He's not verified");
}
```

See also

AddSession, DelSession.

WeakDecryptValue

WeakDecryptValue decrypts a value encrypted using **WeakEncryptValue**.

```
str = WeakDecryptValue(string)
```

Parameters

String is the text to decrypt.

Returns

A string holding the non-decoded value.

Remarks

This is not the secure encoding/decoding used for passwords.

Example

```
Print(WeakDecryptValue("5J%25XiVPd6x8%296eP8iJA3"));
```

Result

```
simon@test.dom
```

See also

WeakEncryptValue

WeakEncryptValue

WeakEncryptValue encrypts a plain text value.

```
str = WeakEncryptValue(string)
```

Parameters

String is the text to encrypt.

Returns

A string holding the non-decoded value.

Remarks

This is not the secure encoding/decoding used for passwords.

Example

```
Print(WeakEncryptValue("simon@test.dom"));
```

Result

5J%25XiVPd6x8%296eP8iJA3

See also

WeakDecryptValue

WildcardFilterMsg

WildcardFilterMsg searches a message for one or more strings.

```
str = WildcardFilterMsg(msg,"string1","string2",.....)
```

Parameters

Msg

The message structure.

Mask

A comma-separated sequence of up to 63 strings.

Returns

For each filter, a count of how many occurrences were found.

Remarks

The strings can contain wildcards although the use of wildcards may significantly impact performance due to the extra work involved.

Example 1

In this example Fred appears in the message five times and "Joe*" eight times. The name Lou is not in the message:

```
WildcardFilterMsg(msg,"Fred ","Joe* ","Lou");
```

Result

5 8 0

Example 2

If you want to filter out messages with .vbs, .exe extensions for example the following might be useful.

```
x = WildCardFilterMsg(msg, "bat", "vbs", "exe");  
if (LSElement(x,1) > 0 || LSElement(x,2) > 0 ...)  
{  
    reject using "Action" or do other stuff  
}
```

See also

FilterMsg

WildcardMatch

WildcardMatch compares two strings, one of which contains wildcards.

tf = WildCardMatch(match, string)

Parameters

Match

The string to look in.

String

A string with wildcards.

Returns

TRUE if the first string matches the string which contains the wildcards, otherwise FALSE.

Example

```
Print(WildcardMatch("JoeBloggs.mbx", "Joe*.mbx"));
```

Result

1

See also

Match

WordWrap

Wordwrap applies word-wrapping to the text specified.

```
tf = WordWrap(text, limit[,inserttext])
```

Parameters

text

The text to be wrapped.

limit

The character position at which to invoke the wrap.

inserttext(optional)

text to be entered at the insert point the default is "\r\n"

Returns

The wrapped text.

Example

```
x = wordwrap("This is a piece of text that needs to be word-wrapped", "20", "<br>");  
print(x);
```

Result

```
This is a piece of  
text that needs to  
be word-wrapped
```

6 Constants

The following constants are used by MML:

Constant	Value
ACCESS_RIGHT_DOMAIN	1
ACCESS_RIGHT_GUEST	6
ACCESS_RIGHT_JUCE	3
ACCESS_RIGHT_LOG	5
ACCESS_RIGHT_SUPPORT	2
ACCESS_RIGHT_SYSTEM	4
AccountTypeAlias	0
AccountTypeAutoResponder	7
AccountTypeDLL	6
AccountTypeForward	2
AccountTypeList	3
AccountTypeListAll	14
AccountTypeListManager	5
AccountTypeMailbox	18
AccountTypeMoved	8
AccountTypeNTDatabase	15
AccountTypeNTMail	1
AccountTypeRobot	9
AccountTypeScript	16
AccountTypeSuperList	4
AccountTypeTemplate	10
AccountTypeTranslate	13
AccountTypeVirtualPOP	11
ALL_SERVICE	9
DomainTypeDLL	16
DomainTypeFull	1
DomainTypePOP	2
DomainTypeRobot	4
DomainTypeVirtual	8
FALSE	0
FILE_APPEND	2
FILE_BINARY	4
FILE_READ	0
FILE_TEXT	3
FILE_WRITE	1
FOLDER_MSG_ANSWERED	0
FOLDER_MSG_DELETED	2
FOLDER_MSG_DIGEST	1

Constant	Value
FOLDER_MSG_DRAFT	3
FOLDER_MSG_FLAGGED	4
FOLDER_MSG_MODERATED	5
FOLDER_MSG_POSTED	6
FOLDER_MSG_READ	7
FOLDER_MSG_RECENT	8
FOLDER_MSG_SEEN	9
FOLDER_MSG_WARNING	10
JOIN_LIST_COMMAND	1
KEY_CUSTOMER	501
KEY_IS_DEMO	402
KEY_PRODUCT_ACTIVE	403
KEY_PRODUCT_NAME	500
KEY_USERS	400
KEY_USERS_PER_LIST	401
LEAVE_LIST_COMMAND	2
LEAVE_REASON_BAD_MESSAGE_LIMIT	3
LEAVE_REASON_BANNED	2
LEAVE_REASON_EXPIRED	0
LEAVE_REASON_MEMBER	1
LINUX	4
LIST_ACCESS_ALLOWED	0
LIST_ACCESS_MODERATE	2
LIST_ACCESS_PASSWORD	3
LIST_ACCESS_SELFONLY	1
LIST_PARM_ACCESS_ANYONE	3
LIST_PARM_ACCESS_FILE_ANYONE	6
LIST_PARM_ACCESS_FILE_ONLY	7
LIST_PARM_ACCESS_MEMBER_ANYONE	4
LIST_PARM_ACCESS_MEMBER_ONLY	5
LIST_PARM_ACCESS_MEMBER_PASSWORD_ANYONE	10
LIST_PARM_ACCESS_MEMBER_PASSWORD_ONLY	11
LIST_PARM_ACCESS_MODERATOR	13
LIST_PARM_ACCESS_NO_ACCESS	1
LIST_PARM_ACCESS_ONLY	2
LIST_PARM_ACCESS_PASSWORD_ANYONE	8
LIST_PARM_ACCESS_PASSWORD_ONLY	9
LIST_PARM_ACCESS_WILDCARD	12
LIST_PARM_ACTIVE	1
LIST_PARM_DO_NOTHING	1
LIST_PARM_INDEX_DIGEST_STR	3

Constant	Value
LIST_PARM_MIME_DIGEST_STR	2
LIST_PARM_ODBC_RO_TYPE	3
LIST_PARM_ODBC_TYPE	2
LIST_PARM_PAUSED	3
LIST_PARM_STOPPED	2
LIST_PARM_SUSPEND_IGNORE	6
LIST_PARM_SUSPEND_TO_ADDRESS	9
LIST_PARM_SUSPEND_TO_MODERATOR	8
LIST_PARM_SUSPEND_TO_OWNER	7
LIST_PARM_TEXT_DIGEST_STR	1
LIST_PARM_TO_ADDRESS	5
LIST_PARM_TO_MODERATOR	3
LIST_PARM_TO_OWNER	2
LIST_PARM_TO_SENDER	4
LIST_PARM_V4_FILE_TYPE	1
LOG_DNS	512
LOG_FAILURE	2
LOG_JUCE	131072
LOG_PROGRESS	4
LOG_PROTOCOL	256
LOG_RETURN	8
LOG_SPAM	4096
LOG_STATS	16
MEMBER_TYPE_BANNED	2
MEMBER_TYPE_BEING_MODERATED	3
MEMBER_TYPE_CONFIRM	4
MEMBER_TYPE_DELETE	7
MEMBER_TYPE_JOIN_EXPIRED	5
MEMBER_TYPE_MEMBER	0
MEMBER_TYPE_OLD	1
MEMBER_TYPE_ZOMBIE	6
MSG_DROP	0
MSG_QUEUE	2
MSG_SEND	1
NTMAIL_INSTALL_DIR	C:\Gordano\
PasswordPolicyDigit	0
PasswordPolicyLetter	1
PasswordPolicySymbol	2
POST_SERVICE	2
PRODUCT_JUCE	4

Constant	Value
PRODUCT_NTLIST	2
PRODUCT_NTMAIL	1
PRODUCT_VIRUS_SCANNER	3
PROXY_PURGE_CREATED	0
PROXY_PURGE_LAST_USED	1
RESUME_LIST_COMMAND	18
RFC822_ADD_FROM	1
RFC822_ADD_TO	2
RFC822_ADD_TO_ALWAYS	3
RFC822_STRICT	0
SMTP_SERVICE	3
SOLARIS	3
SUSPEND_LIST_COMMAND	17
TRUE	1
WIN95	2
WINNT	1
WWW_SERVICE	7

Example

AccountTypeNTDatabase is shown as 15.

TheAccountType is a bitmap and various bits are enabled or disabled depending on what settings are required.

The values are shown in the registry in their decimal form in the user space.

The bits are always worked in binary so to write bit 15 you would use

1000000000000000 binary.

If you convert this to decimal you get a value of 32768

AccountTypeMailbox is set to 18 and is required to enable a mailbox for the user. Writing this in binary gives 100000000000000000 or in decimal a value of 262144.

$$32768 + 262144 = 294192$$

Therefore a value of 294192 indicates that the user is an NT Database user and that the mailbox is enabled for that user.

7 Troubleshooting

This chapter shows to solve problems you may encounter as you learn to write MML scripts. It describes:

- What happens when a script fails — how the failure is reported.
- Advice on debugging your scripts.
- A list of error messages you may see, with advice on how to fix many of the problems.

7.1 Diagnostics

An error will appear in your Web browser. The following is the first part of the output from a failed script:

An error has occurred - Function undefined (1009)

Current location:

2: \Dialup\RuleConfigAdv.mml

1: \Dialup\RuleProcess.mml

Location in \Dialup\RuleConfigAdv.mml:

```
6  MenuTitle = "Connection scheduling";
7
8>  IstRAS = EnumRasEntries();
9  nLen = LStringLength (IstRAS);
10
```

This is followed by a debug list of local, session and form variable values. For example, the form variables section will resemble this:

Form Variables

Variable Name	Value
action	New schedule
HTTP_Accept	image/gif,image/x-xbitmap,image/jpeg, image/pjpeg, */*
HTTP_Accept_Charset	iso-8859-1,* ,utf-8
HTTP_Accept_Language	en
HTTP_Connection	Keep-Alive
HTTP_Content_length	21
HTTP_Content_type	application/x-www-form-urlencoded
HTTP_Host	127.0.0.1:8000
HTTP_REFERER	http://127.0.0.1:8000/Dialup/ Rule.mml?Change Open=Dialup&ChangeTabIndex=1
HTTP_User_Agent	Mozilla/4.01 [en] (WinNT; I)
LOCAL_ADDR	127.0.0.1
LOCAL_HOSTNAME	127.0.0.1
REMOTE_ADDR	127.0.0.1
REMOTE_HOSTNAME	127.0.0.1
SCRIPT_LENGTH	15988
SCRIPT_NAME	/Dialup/RuleProcess.mml
SERVER_NAME	InternetShopper
SERVER_PORT	80
SERVER_PORT_SECURE	0
SERVER_PROTOCOL	HTTP/1.0
SERVER_SOFTWARE	InternetShopper

7.2 List of Script Errors

This section lists all the errors which you may see, with information on how to fix the more complex problems. Any of these errors will report the message "500 Server error indicated".

Syntax error

Unbalanced parentheses
No expression present
Equals sign expected
Not a variable
Parameter error
Semicolon expected
Unbalanced braces
Function undefined
Too many nested function calls
Return without a call
Parentheses expected
While expected
Closing quote expected
Not a string
Too many local variables
Too many function arguments compared with definition
Too few function arguments compared with definition
Strings may not extend over an end-of-line
String is too long
Number is too long
Function requires n parameters
Function, resolved variable or variable name is too long
Hashes unmatched during variable resolution
Domain does not exist
User does not exist in this domain
Cannot add users to this type of domain
Cannot change a constant value
Cannot create a domain of this type
Open Brace expected but not found
Braces not allowed in expression
ODBC Driver reported an error
Variable is not of type MSG
Variable is not of type FILE
File sharing violation
Unexpected end of MML

Bad value
Extra open script command was unexpected
Extra close script command was unexpected
Division by zero
Could not find name of file to include
Too many nested includes
No file found
Variable name cannot be the same as a function name
File type or mode was not recognised
Invalid operator in expression
Variable is not of type FOLDER
The Variable is read only
The end of the message has been found
The system has run out of memory
Too many function definitions
Invalid filename
The function has been already been defined
Operation not allowed by current user

8 FAQs and Examples

8.1 Example Robots

The Gordano Accessory pack contains a number of examples of MML usage including three example robots

Filter robot.

This robot is designed to be used as a user delivery script. Paste the contents of the file filter.mml into the Users\DeliveryScript tab in the Gordano interface. The robot provides examples of how to filter messages and perform various actions on the messages. The filtering looks in the subject of the message or based on it will either copy, transfer, forward, redirect, reply or delete the message.

Template robot.

This robot is designed to be used as a user delivery script. Paste the contents of the file template.mml into the Users\DeliveryScript tab in the Gordano interface. The robot provides functionality similar to the template processing provided by template accounts in SMTP. The following templates are supported:

%subject% The subject of the original message

%date% The date of the original message

%from% The sender of the original message

%to% The recipient of the original message

%header% The header of the original message

%body% The body from the original message. Note that these can be very large.

The script makes every attempt to avoid mail loops by checking for the null account.

Admin robot.

This robot is designed to be used as a user delivery script. Paste the contents of the file filter.mml into the Users\DeliveryScript tab in the Gordano interface. The robot provides "Administration via email" functionality. Addition and deletion of accounts is supported. Access is password protected. The format of the message is:

Password= <admin robot password>

Add fred fredpassword

Add barney barneypassword

Delete wilma

Delete betty

8.2 Example Timed Events

A single example timed event is provided in the Gordano accessory pack. The contents of the file should be pasted into a timed event added via the system>timed events page of the Gordano interface. It will simply zip up a mailbox for a given user and send it to another user.

8.3 Example User Defined GUI

A user definable GUI is included in the Gordano Accessory Pack. To install the GUI these files should be extracted into \mml\usr. The GUI can be accessed by connecting to the GMS server on port 8888. The GUI allows you to send and receive email and to configure an autoresponse. Once installed you can edit the appearance of this GUI to your preference.

8.4 What is an API?

An API or Application Programming Interface, allows programmers to access the functionality of a pre-built software module through well-defined data structures and subroutine calls.

Traditionally the most popular networking APIs have accompanied socket libraries. Berkeley sockets and Windows Sockets (Winsock) APIs have seen widespread use for many years. More recently, Java network APIs such as servlets have also grown in popularity.

Gordano provides many API's to allow you to modify the product behaviour. These include

- The Mail Meta Language, MML (a powerful scripting language) which allows the customisation of functions throughout the Gordano product range.
- SMTP clause DLL.
- An authentication DLL..
- Message DLL.

8.5 What is MML?

MML stands for Mail Meta Language, and is a powerful scripting language designed specifically for the handling of Internet mail. MML is unique to Gordano products.

8.6 What is a script?

A script is a list of commands that can be executed without user interaction. A script language is a simple programming language with which you can write scripts. A script is processed by an interpreter.

Gordano's products support the Mail Meta Language (MML) scripting language. Designed in house by the Gordano developers this scripting language is specifically designed to assist with the processing of Internet mail messages.

There are four types of scripts that can be written in MML, these are:

- **GUI Scripts** - These scripts are used to provide custom interfaces.
- **GMS Anti-Spam Scripts** - These interact with messages passing through the server.
- **Timed Event Scripts** - initiate an event at a specified time. For example to dial up to your ISP or to back up your system.
- **List Messages containing executable MML** – these mix HTML and Text with embedded MML.

8.7 Examples

Can I move all messages over a certain size to a particular account?

If you want to take a copy of all messages over a certain size and send them to a particular account you could use an EOM Script, the following is an example of such a script.

```
if (email\X-MMLScript == " " && email\size > 10240)
{
    action = "-1 ";
    rcpt = "postmaster@test.dom";
}
```

To enter an EOM Script you need to be running GMS Anti-Spam.

Can I copy messages over a certain size to another account?

To do this you can run a delivery script on the user account that you want this to affect, an example of a such a script would be

```
if (email\X-MMLScript == " " && email\size > 10240)
{
    msg = MsgCopy(email);

    if (msg)
    {
        MsgAddHeader(msg, "X-ByWayOf", rcpt);
    }
}
```

```

        MsgAddHeader(msg, "X-MMLScript", "Failed");
        msg\recipient = "postmaster@test.dom";
        MsgClose(msg, MSG_SEND);
    }
}

```

If you wanted to copy the message to another account while discarding the message to the original recipient you need to add an "Action" to the script, as in the following example

```

if (email\X-MMLScript == " " && emailsize > 10240)
{
    action = 1;

    msg = MsgCopy(email);

    if (msg)
    {
        MsgAddHeader(msg, "X-ByWayOf", rcpt);
        MsgAddHeader(msg, "X-MMLScript", "Failed");
        msg\recipient = "postmaster@test.dom";
        MsgClose(msg, MSG_SEND);
    }
}

```

How can I stop the Love Bug virus with GMS?

Please find an MML script below that will trap the "I Love You" virus in SMTP before getting into your system. You do not need the virus scanner package to use this but you will need GMS Anti-Spam.

In the GMS Anti-Spam administration interface go to the "Connect" option, select the "Scripts" tab and add a new script. The script type to add is "End Of Message" and you should paste the MML below directly into the text area. Once you've added the script you should see the script listed. You will need SMTPScripts to be turned on to use this script - this will be enabled when you add the script.

That's all you should need to do. This is better than using the the GMS Anti-Spam content filters as it will happen earlier within SMTP and is guaranteed to always be executed.

The script is self explanatory and could be easily modified to spot the new variants of the virus as they occur or indeed completely different viruses.

-- MML Starts below

```

if (email\X-MMLScript == " ")
{
    if (email\Subject == "ILOVEYOU")

```

```

{
  x = FilterMsg(email, "LOVELETTER", "name=\"LOVE-LETTER-FOR-
YOU.TXT.vbs");

  if (LSElement(x, 1) != 0 && LSElement(x, 2) != 0)
  {
    action = 1;
    parameter = "550 ILOVEYOU virus found in message";
  }
}
}

```

Can I Customise List Postings when the list members are held in a database?

I have a database with several hundred thousand member entries. I want to send out a message but I want different groups of members to get different messages. Specifically I want to easily include in the message different URLs where that URL is made up of details included in that person's database record. E.g. user1,1234,abcd gets the URL
http:\\www.mysite.dom\\1234abcd.htm

This is an extremely good example of just what GMS Communication Server can do.

If you have your information in a database table, it might have columns like:

Domain	User	Item	ItemB
gordano.com	support	A	93
GMS.co.uk	user	asdf	3
		iei	

When an email message is sent to a list with associated with this database table, it might have the following MML code in it:

```

EG>
EG> Hi <# print(user); #>,
EG>
EG> Please click on this URL:
EG> http://mysite.dom/page1/item.htm?id=<# print(ItemA,
"&x=", ItemB); #>
EG>

```

When this is sent to the list, it would end up being customised for each list member according to the fields in the database table:

```
EMAIL1>
EMAIL1> Hi support,
EMAIL1>
EMAIL1> Please click on this URL:
EMAIL1> http://mysite.dom/page1/item.htm?id=asdf&x=93
EMAIL1>

EMAIL2>
EMAIL2> Hi user,
EMAIL2>
EMAIL2> Please click on this URL:
EMAIL2> http://mysite.dom/page1/item.htm?id=iei&x=3
EMAIL2>
```

Can I Customize Mail Outs Using GMS Communication Server or GMS List Server?

I want to use a list that I have set up to send a message to certain members of that list but not to others. Can this be done and if so how?

This is done using MML (Mail Meta Language) and the "Action" variable. For example suppose you want to post to just the list members whose e-mail address begins with "s". To do this you would first have to enable MML by going to Post > processing in the interface and where it says "Execute MML within posted messages" select "Whole Message" from the drop down and click on update. Next you need to post a message containing the following script:

```
<# If (left(RCPT, 1) != "s" ;
Action = 2; #>
Your message goes here.
```

Now only members whose e-mail address begins with "s" will receive the message.

Action Variable Values:

Action = 0 continue the message (default)

Action = 1 retry sending the message later

Action = 2 Never send this message. Delete it from the queue so that we don't try to send it again later. No automatic failure (or bounce) message will be generated.

How do I put scripts into list footers?

All Gordano products have an internal scripting language called "MML" or Mail Meta Language. This language is simple but extremely powerful. GMS Communication Server will allow MML to

be included in the Header/Footer region of messages or anywhere in the message.

First create your list - for example test@test.dom. Proceed to the "GLCom>Post>Processing" page, change "Execute MML within posted messages" to "Headers and Footers" and press update. Press the "EDIT" button to allow you to change the footer of your message. The footer is added to every message as it is posted through GMS Communication Server. In the "footer" section, enter the line:

```
<# print("You joined on ", DateJoined); #>
```

and press UPDATE. Add some users to the list (you could use GMSComm > Manage > Import to do this) and then post a message to the list.

Result: Check the message that has been delivered to one of the list members and you should see the following line added to the bottom (with a different date)!

```
You joined on 1999-09-03 11:20:53
```

Notes: There are several other variables that could have been used, for example:

RCPT	Where the message is being delivered to.
MAIL	Who sent the message.
NumBadMsgs	Number of messages returned to the mailing list by this user.
Name	If the name could be obtained from the email message, the name of the person on the list.
Organization	If the organisation is specified in the email message, this will be the name of the organisation that the person belongs to.

How can I use MML to tell if a message has an attachment?

Is there a way I can detect the presence of an attachment on an incoming message so I can make an auto-responder to tell people to use PUT instead.

You can use the email object to check the content-disposition of the message which will show whether it has an attachment. For example:

```

disp = Trim(email\Content-Disposition);

if (disp != " " && disp != "inline")
{
    if (Instr(disp, "attachment;") || Instr(disp, "filename"))
    {
        /* Do whatever you need */
    }
}

```

Why do messages get through MML filters?

The MML filter is set up to look for multiple occurrences of a specific word and to ban emails where the occurrence of this word exceeds the given number.

Imagine a filter that checks for a word "banned" occurring 5 times in an incoming message. This creates the following script

```

if (email\X-MMLScript == " ")
{
    action = 0;
    if (action == 0)
    {
        x = FilterMsg(email, "banned");
        if (LSElement(x,1) > 4)
        {
            action = "1 ";
            parameter = "550 contains disallowed word";
        }
    }
}

```

If the word "banned" occurs 9 times or less in the message, the script functions correctly, and the message is rejected with the error "550 contains disallowed word".

However if the word "banned" exists greater than 9 times in the message it will pass through the script and will not be rejected.

This is because the MML parser is designed to treat this as a string comparison, as the result of LSElement is a string. Consequently if it finds 10 occurrences of the word, this is equivalent to finding it one time.

To correctly trap and reject these messages you need to use an integer comparison and hence should use the **ToInt** MML function to convert the result of LSElement to an integer which will then be correctly used in comparisons.

This means that the line:

```
if (LSElement(x,1) > 4)
```

in the script above will become

```
if (ToInt(LSElement(x,1)) > 4)
```

Licence Agreements

GORDANO LIMITED SOFTWARE LICENCE AGREEMENT

Copyright © Gordano Ltd, 1995-2015

WARNING: YOU SHOULD CAREFULLY READ THE FOLLOWING TERMS AND CONDITIONS BEFORE USING THIS SOFTWARE PACKAGE. INSTALLING THE SOFTWARE ONTO YOUR COMPUTER INDICATES YOUR ACCEPTANCE OF THESE TERMS AND CONDITIONS. IF YOU DO NOT WISH TO ACCEPT ALL OF THESE TERMS, YOU SHOULD STOP INSTALLING THIS SOFTWARE NOW AND DESTROY ALL COPIES OF THE SOFTWARE AND ALL MANUALS AND OTHER DOCUMENTS SUPPLIED WITH IT.

1 DEFINITIONS

"Agreement" means this Gordano Limited Software Licence Agreement together with all related invoices.

"Company" means the licensee of the Software, being the signatory to this Agreement.

"Gordano" means Gordano Limited.

"Documentation" means any documentation or manuals provided with the Software or provided Online or on storage media containing this text.

"Key" means the activation key.

"Software" means the software computer program, Key and Documentation contained in this package.

"Trial Period" means the period of 28 days from installation of the Software.

2 GRANT OF LICENCE

2.1 Subject to the Company's compliance with the terms of this Agreement, Gordano grants to the Company a non-exclusive, non-transferable licence to use the Software strictly for its own internal business operations only under the terms of this Agreement for the Trial Period and thereafter if a key is purchased from Gordano or its authorised representatives. For the avoidance of doubt, operating the Software outside the Trial Period or without a Key from Gordano (or its representatives) constitutes unlicensed use of the Software and will be a material breach of this Agreement, which would allow Gordano to terminate under clause 8.2.

2.2 This Agreement becomes effective upon the Company signing this Agreement or installing the Software.

2.3 On expiry of the Trial Period and on payment of the fee invoiced by Gordano, the Company will be sent the Key which will activate the Software.

2.4 The Company may use the Software on the number of computers that it has purchased a licence for; a separate license is required for any other computers. The number of licenses purchased by the Company under this Agreement will be stated on the invoice issued by Gordano.

2.5 The Company may make one copy of the Software, strictly for backup or archive purposes only.

2.6 The Company shall be responsible for all use of the Software licenced under this Agreement, including but not limited to any use by its agents, contractors, outsourcers, customers and suppliers, and their compliance with this Agreement.

2.7 The Company agrees to maintain accurate and adequate records relating to its use of the Software and compliance with this Agreement. The Company agrees to permit Gordano to audit the Company in relation to its use of the Software and compliance with the terms of this Agreement. The Company shall provide Gordano with reasonable assistance and access to information in the course of any such audit, and the Company agrees that Gordano may report the audit results to its licensors. Each party shall be responsible for its own costs in relation to any such audit.

2.8 In the event that the Software contains source code from a licensor of Gordano, that source code shall also be governed by the terms of this Agreement.

3 OWNERSHIP OF THE SOFTWARE

3.1 Gordano and its licensors own all title and proprietary rights to the Software and all copies thereof and all rights therein, including without limitation all copyright, patents, know-how, trade secrets, trade marks or names and database rights. All such rights shall remain vested in Gordano and its licensors. The provision of the Software to you does not grant, and you do not receive, any rights under any Microsoft intellectual property with respect to any device or software that you use to access the Software.

3.2 The Company undertakes and agrees as follows:

- (a) it may NOT make or permit others to make any copies of the Software except for one backup copy.
- (b) it may NOT reverse engineer, disassemble, decompile the Software or attempt to reconstruct, identify or discover any source code except as expressly permissible by law.
- (c) it may NOT modify, adapt or translate the Software or incorporate the Software, in whole or in part in any other product or software or permit others to do so without express, written consent of Gordano.
- (d) it may NOT disclose, provide or otherwise make available in any form the Software, its functionality or any portion thereof, to any third party other than its employees without the prior written consent of Gordano.
- (e) it may NOT remove any copyright, trademark, proprietary rights, disclaimer or warning notice included on or embedded in any part of the Software and the Company agrees to diligently reproduce all copyright notice(s) and other proprietary notices of Gordano on any authorised copy of the Software.
- (f) it may NOT assign, sell, transfer (except for temporary transfer in the event of computer malfunction), licence, sub-licence, rent, timeshare, lease or otherwise redistribute the Software or its functionality to any third party without the written permission of Gordano.
- (g) it may NOT use the Documentation for any purpose other than to support its use of the Software.
- (h) it accepts that from time to time, the Software will send a message containing details of the Key or Keys installed to Gordano and it agrees not to interfere with the delivery of this message.
- (i) it accepts, that Gordano may receive error messages from the Software installed on the Company's system in the event that the Software fails for some reason (and that the Company has the option to turn this off).
- (j) it agrees to stop using all previous version of the Software immediately following an upgrade.
- (k) it may NOT use the Software for any subscription service, hosting or outsourcing.
- (l) it may NOT publish any results of benchmark tests run on the programs.
- (m) if appropriate, it must comply with all relevant import and export laws to ensure that the Software or anything directly produced using the Software are not exported directly or indirectly contrary to applicable laws.
- (n) it agrees that any third party technology that may be appropriate or necessary for use with some or all of the Software that is notified to the Company (whether via the Documentation or otherwise) shall not be licensed to the Company under this Agreement, but may be licensed as stated in the Documentation or as otherwise notified to the Company.
- (o) The Company shall ensure that its customers and/or employees (and any other

persons) that use the Software agree to and are bound by the following condition on their right to access and use the Software: "The provision of the Software to you does not grant, and you do not receive, any rights under any Microsoft intellectual property with respect to any device or software that you use to access the Software."

3.3 No distribution licence or other rights are provided to the Company under this agreement.

3.4 The Software may utilise Microsoft® Exchange ActiveSync, and the use of Microsoft® Exchange ActiveSync is limited to internal use as part of hosting the Software for the sole purpose of providing access by Microsoft® approved devices to email accounts of employees or customers of the Company maintained by the Software.

The provisions of clauses 3, 4, 6, and 7 shall survive termination of this Agreement.

4 CONFIDENTIALITY

4.1 The Company undertakes to treat as confidential and keep secret all information contained or embodied in the Software and Documentation supplied by Gordano.

5 ANTI-VIRUS

5.1 Gordano does not warrant that the Software is free from all known viruses and the Company shall assume responsibility to take appropriate steps to ensure that the Software is virus free and that the running of the Software will not damage or interfere with the computer system on which the Software is used or any data or software which may be used or stored on its computer system.

6 WARRANTY AND DISCLAIMER

6.1 The Company acknowledges that software in general is not error free and agrees that the existence of such errors in the Software shall not constitute a breach of this Agreement.

6.2 The Company further acknowledges that the Software has not been developed to meet its specific individual requirements and that it is the Company's responsibility to ensure that any use of the Software or the information contained on it is suitable for its specific individual requirements.

6.3 THIS SOFTWARE IS PROVIDED 'AS IS'. GORDANO WARRANTS THAT THE SOFTWARE WILL SUBSTANTIALLY COMPLY WITH THE SPECIFICATIONS SET OUT IN THE DOCUMENTATION. EXCEPT AS STATED HEREIN AND TO THE EXTENT PERMITTED BY LAW THE SOFTWARE IS PROVIDED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, STATUTORY OR OTHERWISE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY AND FITNESS FOR A PARTICULAR PURPOSE. GORDANO DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET THE COMPANY'S REQUIREMENTS OR THAT THE OPERATION OF THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE.

6.4 Gordano does not represent or warrant that the Software furnished hereunder is free of infringement of any third party patents, copyrights, other intellectual property rights or trade secrets. The Company waives any right to indemnification or other relief from Gordano should the Software be found to be defective or to infringe any right of any third party.

6.5 Nothing in this Agreement shall exclude or limit the liability of Gordano for death or personal injury

caused by its negligence or for any other liability which cannot by law be excluded. GORDANO'S SOLE LIABILITY TO THE COMPANY FOR ANY CLAIM, DEMAND OR CAUSE OR ACTION WHATSOEVER, AND REGARDLESS OF FORM OF ACTION, WHETHER IN CONTRACT OR TORT, SHALL BE LIMITED TO REPLACEMENT OF THE PRODUCT OR REFUND OF THE LICENCE FEE PAID FOR THE SOFTWARE. IN NO EVENT SHALL GORDANO OR ITS LICENSORS BE LIABLE TO THE COMPANY FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL OR PUNITIVE DAMAGES WHATSOEVER, INCLUDING BUT NOT LIMITED TO LOSS OF ANTICIPATED SAVINGS, LOSS OF REVENUES, LOSS OF PROFIT, LOSS OF BUSINESS, LOSS OF DATA OR DATA USE OR ECONOMIC LOSS OF ANY KIND.

7 LIMIT OF LIABILITY

7.1 In the event that any exclusion or limitation in clause 6 above is held to be invalid for any reason and Gordano becomes liable for loss or damage that may lawfully be limited, such liability shall be limited to the sum equivalent to a multiple of 3 (three) times the total annual fee paid by the Company to Gordano for the licence of the Software.

8 TERMINATION OF LICENCE

8.1 Save in the event of any unlicensed use of the Software when the terms of this agreement shall remain in full force and effect, the Company may terminate this Agreement, at any time, by destroying or returning all copies of the Software.

8.2 Gordano may terminate this Agreement by written notice to the Company if the Company is in default of any terms or conditions of this Agreement or if the Company enters into any form of insolvency including without limitation liquidation, receivership, voluntary arrangement, administration or is unable to pay its debts as they fall due.

8.3 On termination of this Agreement the Company agrees to discontinue all use of the Software and destroy all copies of the Software in any form in its possession or control, and if requested by Gordano certify in writing that such action has been taken. The Company shall not be entitled to any refund of any monies or other consideration paid by it.

9 SUPPORT

9.1 Gordano shall provide support for the first 28 days from your first contact with Gordano or its representatives. First contact means the Company's representative's first telephone call to Gordano, registration on the Gordano website, or installation of the trial software from our website, whichever is the earlier.

9.2 On expiry of this 28 days the Company shall have the option of purchasing support services from Gordano under the terms of the Support Agreement.

10 MAINTENANCE (Software Updates)

10.1 Gordano shall provide maintenance services in the form of updates to the Software for the duration of the Software's licence term, commencing on the expiry of the Trial Period and on the Company's receipt of the Key. Thereafter, the Company shall have the option of renewing annual maintenance services (Software updates) from Gordano.

10.2 Maintenance services shall comprise of the provision of new versions of the Software only as and when they become available, and no other maintenance services or assistance is included.

11 GENERAL

11.1 If any provision of this Agreement is determined to be invalid or unenforceable, by any court of competent jurisdiction it shall be deemed to be omitted and the remaining provisions shall continue in full force and effect.

11.2 Gordano's waiver of any right shall not constitute a waiver of that right in the future.

11.3 This Agreement shall be governed and construed in accordance with the laws of England and both parties submit to the exclusive jurisdiction of the English courts, save in respect of enforcement where the jurisdiction shall be non-exclusive.

11.4 This Agreement constitutes the entire understanding between the parties with respect to the subject matter hereof. The Company agrees that any of Gordano's licensors that are associated with the Software shall be a third party beneficiary of this Agreement. All prior agreements, representations, statements and undertakings, oral or written, between the Company and Gordano are hereby expressly superseded and canceled.

11.5 All notices under this Agreement shall be in writing and shall be given by registered or certified mail to the following address: Gordano Ltd, 1 Yeo Bank Business Park, Kenn Road, Kenn, Clevedon, North Somerset,
BS21 6UW, UK.

GORDANO LIMITED SUPPORT AGREEMENT

WARNING: YOU SHOULD CAREFULLY READ THE FOLLOWING TERMS AND CONDITIONS. BY REGISTERING FOR SUPPORT SERVICES TO BE PROVIDED BY GORDANO YOU ARE ACCEPTING THESE TERMS AND CONDITIONS. IF YOU DO NOT WISH TO ACCEPT ALL OF THESE TERMS YOU SHOULD IMMEDIATELY NOTIFY GORDANO AND ANY SUPPORT FEE YOU MAY HAVE PAID WILL BE REFUNDED FOR THE OUTSTANDING CONTRACT TERM.

1 DEFINITIONS

"Business Days" means weekdays excluding weekends, and UK Bank and Public Holidays and Gordano's training days (which will be notified to the Company in advance and in any case will not be more than 3 (three) days in any one calendar year).

"Company" means the licensee of the Software.

"Gordano" means Gordano Limited.

"Key" means the activation key for the Software or Support Service.

"Software" means the software computer program and documentation licensed to the Company from Gordano.

"Software Licence" means the software licence granting the Company a non-exclusive, nontransferable licence of the Software.

"Support Fee" means the fees payable for the Support Service, which shall be in accordance with Gordano's current price list as amended from time to time.

"Support Agreement" means this Gordano Limited Support Agreement.

"Support Service" means the support services provided by Gordano in relation to the Software and as detailed in clause 3 of this Support Agreement.

2 GRANT

2.1 This Support Agreement is for the provision of Gordano's Support Service in respect of the current version of the Software for the term of your subscription to the Support Service commencing from the date of the commencement of your subscription for the Support Service.

2.2 If further products are licensed from Gordano during the lifetime of this Agreement a "top-up" fee may be added to extend this Support Agreement to cover the additional products at the time of their purchase.

2.3 This Support Agreement becomes effective on the date you pay for the Support Service.

2.4 Customers may register as users on the helpdesk at <https://helpdesk.gordano.com> however this is not required in order to receive support.

3 SUPPORT SERVICES

3.1 Gordano shall provide the Company with the following Support Service:

- (a) telephone support for the Software (currently on +44 (0)1275 340151):
 - (i) between the hours of 0900 to 17:00 or 14:00 to 2200 hours UK Time; or
 - (ii) between the hours of 0900 to 2200 UK Time; on all Business Days or
 - (iii) for 24x7 cover; telephone support shall be provided at all hours on all days
- (b) email support for the Software at support@gordano.com or helpdesk@gordano.com.

3.2 Messages sent to and Support calls made to Gordano will be processed automatically and

assigned a ticket ID. Gordano will send confirmation of these details to the creator of the ticket.

3.3 All Support Services for the Software will be provided in the English language only.

4 EXCLUDED SERVICES

The Support Service supplied under this Agreement shall not include the provision of Support Service in respect of:

- (a) any version of the Software which is more than 24 months past its release date, except at the discretion of a support engineer or the management of Gordano Ltd;
- (b) any products or services which are not the Software or its components;
- (c) training in the use of the Software;
- (d) any development services;
- (e) defects or errors resulting from any modifications or enhancements of the Software made by any person other than Gordano;
- (f) use of the Software other than in accordance with the documentation or operator error;
- (g) virus protection or bug fixes except in exceptional circumstances as advised by Gordano, for example, when the system has been compromised by some external force and there is no available workaround; or
- (h) any circumstances beyond the reasonable control of Gordano, including (but not limited to) any act of God, fire, flood, war, act of violence or any other similar occurrence or failure or reduced performance of telecommunications networks or the internet.

5 COMPANY OBLIGATIONS

5.1 The Company agrees and undertakes:

- (a) to ensure that the Software is used only in accordance with the documentation or advice from Gordano, by competent trained employees only or by persons under their supervision;
- (b) not to alter or modify the Software in any way whatever nor permit the Software to be combined with any other programs to form a combined work;
- (c) not to request, permit or authorise anyone other than Gordano or its nominated third parties to provide any support services in respect of the Software;
- (d) to co-operate fully with Gordano's personnel in the diagnosis of any error or defect in the Software;
- (e) if necessary, to make available to Gordano free of charge all information facilities and services reasonably required by Gordano to enable Gordano to provide the support services;
- (f) to provide such telecommunication facilities as are reasonably required by Gordano for testing and diagnostic purposes.

6 SUPPORT FEES

In consideration of the Support Services the Company shall pay the Support Fee in advance to Gordano

7 TERMINATION

Gordano may terminate this Support Agreement by written notice to the Company if the

Company is in default of any terms or conditions of this Support Agreement by written notice to the Company or if the Company enters into any form of insolvency including without limitation liquidation, receivership, voluntary arrangement, administration or are unable to pay its debts as they fall due.

8 LIABILITY

Gordano's sole liability to the Company for any claim, demand, cause or action whatsoever, and regardless of form of action, whether in contract or tort, including negligence, shall be limited, at Gordano's sole option, to refund of the purchase price, re-performance of the Support Service or an extension to the length of the Support Service to be provided. In no event shall Gordano be liable for recovery of any special, indirect, incidental, or consequential damages, even if Gordano has been advised of the possibility of such damages, including but not limited to lost profits, lost savings, lost revenues, lost business, lost data or economic loss of any kind, or for any claim by any third party.

9 LIMIT OF LIABILITY

In the event that any exclusion or limitation in clause 8 above is held to be invalid for any reason and Gordano becomes liable for loss or damage that may lawfully be limited, such liability shall be limited to the sum equivalent to a multiple of three times the Support Fees paid by the Company to Gordano.

10 GENERAL

10.1 If any provision of this Support Agreement is determined to be invalid or unenforceable, by any court of competent jurisdiction it shall be deemed to be omitted and the remaining provisions shall continue in full force and effect.

10.2 Gordano's waiver of any right shall not constitute a waiver of that right in the future.

10.3 This Support Agreement shall be governed and construed in accordance with the laws of England and both parties submit to the exclusive jurisdiction of the English courts, save in respect of enforcement where the jurisdiction shall be non-exclusive.

10.4 This Support Agreement constitutes the entire understanding between the parties with respect to the subject matter hereof and all prior agreements, representations, statements and undertakings, oral or written, are hereby expressly superseded and cancelled.

10.5 All notices in connection with this Agreement shall be in writing and shall be given by registered or certified mail to the following address: Gordano Ltd, 1 Yeo Bank Business Park, Kenn, Kenn Road, Clevedon, North Somerset, BS21 6UW, UK.

© 2015. Gordano Limited. All rights reserved.

LICENCE AGREEMENT MySQL AB

MySQL AB, Bangårdsgatan 8, 753 20 Uppsala, SWEDEN

1. License Grant. Customer is granted a limited, non-exclusive, non-transferable license to run one copy of the object code version of the Licensed Software on one machine or instrument solely as integrated with, and for running and extracting data from, a Licensee Application. Use shall be limited to internal business purposes in accordance with these license terms. If the Integrated Product is licensed for concurrent or network use, Customer may not allow more than the maximum number of authorized users to access and use the Licensed Software concurrently.

2. License Restrictions. Customer may make copies of the Licensed Software only for backup and archival purposes. Customer shall not:

- (a) copy the Licensed Software onto any public or distributed networks
- (b) use the Licensed Software as a general SQL server, as a stand alone application or with applications other than Licensee Applications under this license;
- (c) change any proprietary rights notices which appear in the Licensed Software; or
- (d) modify the Licensed Software.

3. Ownership. MySQL AB and its third party suppliers retain all right, title and interest in the Licensed Software and all copies thereof, including all copyright and other intellectual property rights. MySQL AB may protect its rights in the Licensed Software in the event of any violation of this EULA.

4. Transfer. Customer may transfer the license granted herein provided that it complies with any transfer terms imposed by Licensee and delivers all copies of the Licensed Software to the transferee along with this EULA. The transferee must accept the terms and conditions of this EULA as a condition to any transfer. Customer's license to use the Licensed Software will terminate upon transfer. Customer must comply with all applicable export laws and regulations.

5. Termination. Upon termination of this license, Customer must immediately destroy all copies of the Licensed Software.

The MD5 Message-Digest Algorithm

The MD5 Message-Digest Algorithm used in NTMail is copyright (c) 1992-2, RSA Data Security, Inc. Created 1991. All rights reserved.

License to copy and use this software is granted provided that it is identified as the "RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing this software or this function.

License is also granted to make and use derivative works provided that such works are identified as "derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either the merchantability of this software or the suitability of this software for any particular purpose. It is provided "as is" without express or implied warranty of any kind.

These notices must be retained in any copies of any part of this documentation and/or software.

jQuery MIT License

Copyright (c) 2008 John Resig, <http://jquery.com/>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Installation and Contact Information

For installation you need the following information. Keep a note of the values you used here in case you need to quote them to support.

Your domain name

Your computer's IP address (if static).

Telephone number of ISP's computer.

Your account user name at the ISP and its password.

To contact Gordano Ltd. :

Support

- Email: support@gordano.com

Sales

- Email: sales@gordano.com
- Tel: +44 1275 345100
- Fax: +44 1275 340056
- Unit 1, Yeo Bank Business Park, Kenn Road, Clevedon, North Somerset, BS21 6UW.

Gordano Limited

Unit 1, Yeo Bank Business Park, Kenn Road, Clevedon,
North Somerset, BS21 6UW. UK

<http://www.gordano.com>

