

# Miva Script Guide

---

Miva Script version: 4.0x  
Guide Revision 1.0



**Miva Corporation**

5060 Santa Fe Street  
San Diego, CA 92109

Telephone: 858-490-2570

Telefax: 858-731-4200

<http://www.miva.com>

This document and the software described by this document are copyright 2002 by Miva Corporation. All rights reserved. Use of the software described herein may only be done in accordance with the License Agreement provided with the software. This document may not be reproduced in full or partial form except for the purpose of using the software described herein in accordance with the License Agreement provided with the software. Information in this document is subject to change without notice. Companies, names and data used in the examples herein are fictitious unless otherwise noted.

Miva is a registered trademark of Miva Corporation. Miva Script, Miva Script Compiler, Miva Order, Miva Merchant, Miva Mia, Miva Empresa, the Miva "blades" logo, Miva Engine, and Miva Virtual Machine, are trademarks of Miva Corporation. Windows is the registered trademark of Microsoft Corporation. All other trademarks are the property of their respective owners. This document was developed and produced in San Diego, CA, USA.

MIVA CORPORATION WILL NOT BE LIABLE FOR (A) ANY BUG, ERROR, OMISSION, DEFECT, DEFICIENCY, OR NONCONFORMITY IN MERCHANT OR THIS DOCUMENTATION; (B) IMPLIED MERCHANTABILITY OF FITNESS FOR A PARTICULAR PURPOSE; (C) IMPLIED WARRANTY RELATING TO COURSE OF DEALING, OR USAGE OF TRADE OR ANY OTHER IMPLIED WARRANTY WHATSOEVER; (D) CLAIM OF INFRINGEMENT; (E) CLAIM IN TORT, WHETHER OR NOT ARISING IN WHOLE OR PART FROM MIVA CORPORATION'S FAULT, NEGLIGENCE, STRICT LIABILITY, OR PRODUCT LIABILITY, OR (F) CLAIM FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES, OR LOSS OF DATA, REVENUE, LICENSEES GOODWILL, OR USE. IN NO CASE SHALL MIVA CORPORATION'S LIABILITY EXCEED THE PRICE THAT LICENSEE PAID FOR MERCHANT.

MS1005-01 (Rev. 1.0) Miva Script 4.0x

# Miva Script Guide

---

## Table of Contents

<b>Miva Script Guide</b> .....	<b>i</b>
<b>Miva Script Tags List</b> .....	<b>iv</b>
<b>Introduction</b> .....	<b>1</b>
<b>Expressions</b> .....	<b>1</b>
<b>Operators</b> .....	<b>2</b>
<b>Variables</b> .....	<b>6</b>
<b>Built-In System Variables</b> .....	<b>7</b>
<b>Functions</b> .....	<b>11</b>
<b>Built-in Functions</b> .....	<b>11</b>
<b>Assignment and Evaluation Tags</b> .....	<b>20</b>
<b>Configuration and Function Tags</b> .....	<b>21</b>
<b>Localization Tags</b> .....	<b>22</b>
<b>Database Tags</b> .....	<b>23</b>
<b>Email Tags</b> .....	<b>30</b>
<b>External File Tags</b> .....	<b>31</b>
<b>Flow Control Tags</b> .....	<b>32</b>
<b>HTTP and Commerce Server Tags</b> .....	<b>33</b>

# Miva Script Tags List

---

<MIVA> .....	21	<MvIMPORT> .....	31
<MvADD> .....	23	<MvIMPORTSTOP> .....	31
<MvASSIGN> .....	20	<MvLOCALIZED> .....	22
<MvASSIGNARRAY> .....	20	<MvLOCALIZED-TEXT> ...	23
<MvCALL> .....	33	<MvLOCALIZED-TOKEN> ..	23
<MvCALLSTOP> .....	35	<MvLOCKFILE> .....	31
<MvCLOSE> .....	23	<MvMAKEINDEX> .....	28
<MvCLOSEVIEW> .....	24	<MvMEMBER> .....	21
<MvCOMMERCE> .....	35	<MvOPEN> .....	25
<MvCOMMERCESTOP> ...	35	<MvOPENVIEW> .....	25
<MvCREATE> .....	24	<MvPACK> .....	26
<MvDELETE> .....	24	<MvPOP> .....	30
<MvDIMENSION> .....	20	<MvPOPDELETE> .....	30
<MvDO> .....	31	<MvPOPSTOP> .....	30
<MvELSE> .....	32	<MvPRIMARY> .....	26
<MvELSEIF> .....	32	<MvQUERY> .....	26
<MvEVAL> .....	21	<MvREINDEX> .....	28
<MvEXIT> .....	32	<MvREVEALSTRUCTURE> ..	26
<MvEXPORT> .....	31	<MvSETINDEX> .....	28
<MvFILTER> .....	24	<MvSKIP> .....	27
<MvFIND> .....	25	<MvSMTP> .....	30
<MvFUNCTION> .....	22	<MvUNDELETE> .....	27
<MvFUNCTION> .....	22	<MvUPDATE> .....	27
<MvGO> .....	25	<MvWHILE> .....	33
<MvHIDE> .....	22	<MvWHILESTOP> .....	33
<MvIF> .....	32		

## Introduction

Miva Script programs are HTML documents that also contain tags (commands) from the Miva Script programming language. Miva Script is a server-side scripting language that is implemented by the Miva Virtual Machine, rather than by the browser. By contrast, a client-side language, such as JavaScript, is implemented by the browser. Miva Script programs are called *scripts*, *active documents*, or simply *documents*. Miva Script tags correspond to typical programming language constructs such as assignment statements, conditional expressions, loops, and input/output statements, as well as Miva Script's special database, mail, commerce, configuration, and external database functionality.

Miva Script tags are XML-based: they have the same format as HTML tags; the element names (for example, <MvASSIGN>) indicate their function; and the attributes specify values that the tags operate on. Miva Script tags can be freely mixed with HTML tags.

Miva Script programs are compiled by the Miva Script Compiler, then run under the Miva Virtual Machine. (Either the Windows desktop version, Miva Mia, or the server version, Miva Empresa.)

When a browser requests a Miva Script document (.mvc), the Miva Virtual Machine executes the contents of the document before passing it on to the browser.

---

## About This Guide

This Miva Script Guide contains the names and syntax for tags, variables, expressions, literals, operators, arrays and structures, functions, and system variables, with only brief descriptions, if any. For detailed information, examples, full descriptions, and explanations, see the *Miva Script Reference Guide* or the *Miva Script User's Guide* at <http://www.miva.com/docs/api>.

## Expressions

Expressions can contain variables, literal values (text or numbers), functions, and operators, which indicate how the other components in the expression are to be evaluated.

- See “[Variables](#)” and “[Built-In System Variables](#)”
- See “[Operators](#)”.
- See “[Functions](#)” and “[Built-in Functions](#)”

In general, Miva Script expressions are used to:

- create a new value. (See “[<MvASSIGN>](#)”.)
- evaluate an expression's logical "truthfulness."

For example, an expression can evaluate a condition, and take an action based on the results. See “[Flow Control Tags](#).”

- Any expression that is equal to 0 (zero) or a null string is considered false.
- All others are considered true. True is often represented by 1 (one).

---

## Expression Syntax

Expressions are enclosed within double quotes, and when it is an attribute value, curly brackets must immediately precede and follow the quotes.

```
<MvASSIGN NAME="field" VALUE="{field + 10}">
```

### Spaces

Inside an expression, spaces are not significant (that is, they do not affect how the expression is evaluated) except in the following ways:

- If an operator contains characters that can also occur in a variable name, then it must be separated from the other components by one or more spaces.
- Spaces inside literal strings are significant.

### Literal

An expression consisting of only a literal string does not require curly brackets. If a literal string is inside curly brackets, it must be surrounded by single quotes. For example:

```
<MvASSIGN NAME="boss" VALUE="{ 'Fred ' $ 'Flintstone' }">
```

### Null

You should represent a null string in an expression as two single quotes ( ' ' ):

```
<MvIF EXPR="{entry EQ ' '}">
```

The expression {0 EQ ' '} ("does zero equal the null string?") returns 1 (true).

---

**Note:** To enter a double quote within an expression, use the `asciichar()` function. (See [“Other String Functions.”](#))

---

See the *Miva Script Reference Guide* for detailed descriptions and examples of expressions containing literal text, numbers, variables, functions, and operators.

## Operators

---

### Order of Precedence

You can also override the built-in precedence by surrounding sub-expressions with parentheses, '(' and ')'.

1. Sub-expressions inside parentheses, '(' and ')', are evaluated first.
2. Each operator is assigned a precedence, and if there is a choice of which operator to evaluate first, the operator with higher precedence is evaluated.
3. If two operators have the same precedence, the left most one is evaluated first.

### Operator Precedence

From highest to lowest Miva Script Operators:

- NOT
- ROUND, CRYPT, MOD, SUBSTR, POW
- /, \*
- +, -, \$
- IN, CIN, EIN, ECIN, EQ, NE, GE, LE, LT, GT
- AND, OR

---

## Arithmetical Operators

A minus sign can precede a literal number to make it negative (for example, -3.14), but, otherwise, must be used strictly as a binary operator: `{ -x }` evaluates to the literal string `'-x'`; to express the negative of a value, use `{ -1*x }` or `{ 0-x }`. Using `+`, `/`, or `*` as a unary operator results in an expression error.

Operator	Name	Description
<code>+</code>	Addition	<code>expr1 + expr2</code>
<code>-</code>	Subtraction	<code>expr1 - expr2</code>
<code>*</code>	Multiplication	<code>expr1 * expr2</code>
<code>/</code>	Division	<code>expr1 / expr2</code>
<code>POW</code>	<b>POW</b> er	<code>expr1 POW expr2</code> Raise <code>expr1</code> to the power <code>expr2</code> <b>Example:</b> <code>3 POW 4</code> is <code>3<sup>4</sup></code> , or 81. See also the built-in function <code>power()</code> .
<code>MOD</code>	<b>MOD</b> ulu	<code>expr1 MOD expr2</code> Returns the integer remainder from <code>expr1/expr2</code> <b>Example:</b> <code>26 MOD 7</code> is 5. See also the built-in function <code>fmod()</code> , which returns floating point remainders.
<code>ROUND</code>	Number rounding	<code>number ROUND places</code> Rounds <code>number</code> up or down to <code>places</code> decimal places. <b>Example:</b> <code>123.45676 ROUND 2</code> is <code>123.46</code> See also the built-in function.

---

## Comparison Operators

These operators are used to compare two numbers or two text strings. In text string comparisons, lowercase letters are considered to be greater than uppercase letters. Two strings are equal only if the case (upper or lower) matches letter-by-letter. These operators all give a value of 1 (true) or 0 (false).

Operator	Name	Description
<code>GT</code>	<b>Greater Than</b>	<code>expr_a GT expr_b</code> Tests whether <code>expr_a</code> is greater than <code>expr_b</code>
<code>LT</code>	<b>Less Than</b>	<code>expr_a LT expr_b</code> Tests whether <code>expr_a</code> is less than <code>expr_b</code>
<code>EQ</code>	<b>EQ</b> ual To	<code>expr_a EQ expr_b</code> Tests whether <code>expr_a</code> is equal to <code>expr_b</code> Two strings are equal only if the case (upper or lower) matches letter-by-letter.
<code>NE</code>	<b>Not Equal To</b>	<code>expr_a NE expr_b</code> Tests whether <code>expr_a</code> is not equal to <code>expr_b</code>
<code>GE</code>	<b>Greater Than or Equal To</b>	<code>expr_a GE expr_b</code> Tests whether <code>expr_a</code> is greater than or equal to <code>expr_b</code>
<code>LE</code>	<b>Less Than or Equal To</b>	<code>expr_a LE expr_b</code> Tests whether <code>expr_a</code> is less than or equal to <code>expr_b</code>

---

**Note:** The expression `{ 0 EQ '' }` ("does zero equal the null string?") returns 1 (true).

---

---

## Logical Operators

These operators are used with 'true' or 'false' expressions. For example:

```
<MvIF EXPR="{age GE 17 AND age LT 80}">
```

This expression is true if *both* expressions '{age GE 17}' and '{age LT 80}' are true.

Operator	Name	Description
NOT	Logical <b>NOT</b>	NOT <i>expr</i> Returns the opposite of <i>expr</i> (if <i>expr</i> is false, <i>NOT expr</i> is true, and vice versa). Notice that this operator is unary: it acts on one expression, not two.
AND	Logical <b>AND</b>	<i>expr_a</i> AND <i>expr_b</i> This expression is true if both <i>expr_a</i> and <i>expr_b</i> are true.
OR	Logical <b>OR</b>	<i>expr_a</i> OR <i>expr_b</i> This expression is true if either of <i>expr_a</i> and <i>expr_b</i> is true.

---

## Text String Operators

Miva Script also supports a number of built-in string functions.

Operator	Name	Description
\$	Concatenate strings	<i>expr_a</i> \$ <i>expr_b</i> Concatenates (joins) the strings <i>expr_a</i> and <i>expr_b</i> together. For example, {'fred' \$ 'flintstone'} would result in 'fredflintstone'.
IN / CIN	Beginning position	<i>expr_a</i> IN <i>expr_b</i> <i>expr_a</i> CIN <i>expr_b</i> Returns the <i>beginning</i> position of <i>expr_a</i> contained in <i>expr_b</i> . For example, {'da' IN 'canada'} returns 5, because 'da' begins at the fifth letter in 'canada'. IN is case-sensitive: it will find matches only if the matched sub-string in <i>expr_b</i> has the same case, letter-by-letter, as <i>expr_a</i> . Use the CIN operator if you want to make your comparison case-insensitive. Notice that if you use literal strings in this expression you have to surround them with <i>single</i> quotes.
EIN / ECIN	End position	<i>expr_a</i> EIN <i>expr_b</i> <i>expr_a</i> ECIN <i>expr_b</i> Returns the <i>end</i> position of <i>expr_a</i> contained in <i>expr_b</i> . For example, {'dia' IN 'canadian'} returns 7, because 'dia' ends at the seventh letter in 'canadian'. EIN/ECIN returns 0 when the left operand is NULL. This fixes backward compatibility with Miva v3.57 and earlier "Htmlscript" versions. <b>ECIN</b> is the case-insensitive version of <b>EIN</b> .
CRYPT	Encrypt a string	<i>plaintext</i> CRYPT <i>key</i> Performs a one-way encryption, similar to that provided with the UNIX <i>crypt</i> command. The string <i>plaintext</i> is encrypted using the string <i>key</i> (this string is sometimes called a 'salt'). CRYPT always yields the same result when applied to a particular <i>plaintext</i> and <i>key</i> . FreeBSD applications can take advantage of crypt(3) features.

---

## Bitwise Operators

These operators act on numbers at the binary or 'bit' level.

Operator	Name	Description
<b>BITAND</b>	Bitwise AND	<code>expr_a BITAND expr_b</code> Perform a logical AND on the bits of <code>expr_a</code> and <code>expr_b</code> .
<b>BITOR</b>	Bitwise OR	<code>expr_a BITOR expr_b</code> Perform a logical OR on the bits of <code>expr_a</code> and <code>expr_b</code> .
<b>BITXOR</b>	Bitwise Exclusive OR	<code>expr_a BITXOR expr_b</code> Perform a logical exclusive OR on the bits of <code>expr_a</code> and <code>expr_b</code> (return the number whose bits are equal to 1 in either, but not both, of the original numbers).
<b>BITOC</b>	Bitwise ones complement	<code>BITOC expr</code> Flip the bits of <code>expr</code> (including the 'sign bit'). This operator is unary: it acts on one number, not two. Example: <code>{BITOC 9}</code> . 9 in binary form is '1001'; including the 'sign bit' (left most bit), which indicates that the number is positive, it is '01001'. Flipping these bits gives '10110'. Since the sign bit is now '1', the number is negative. According to the rules of binary arithmetic, '0110' interpreted as a negative number is '-10'.
<b>BITSL</b>	Bitwise shift-left	<code>expr_a BITSL expr_b</code> Shift the bits of <code>expr_a</code> to the left by <code>expr_b</code> places. The leftmost bits are lost, and the rightmost bits are replaced by zeroes. ' <code>{23 BITSL 2}</code> ' is interpreted as follows: 23 is 00010111 in binary form; shifting these bits left two places gives 01011100, or 92 in decimal format.
<b>BITSR</b>	Bitwise shift-right	<code>expr_a BITSR expr_b</code> Shift the bits of <code>expr_a</code> to the right by <code>expr_b</code> places. The rightmost bits are lost, and the leftmost bits are replaced by zeroes.

---

## Variables

---

### Prefixes and Scope

Prefixes in the variable name define its scope in the current program. The variable scopes available in Miva Script are:

- Local  
(prefix *local* or *l*): Scope is only inside the <MvFUNCTION> block.
- Database  
(*dbname.database.var* or *dbname.d.var*): Scope is named database.
- System  
(prefix *system.varname* or *s.varname*): Used when a built-in system variable or a variable that is generated by an <MvSMTP>, <MvPOP>, or <MvCALL> loop is named.
- Global  
(prefix *global* or *g*): Scope is the entire program

If scope is not defined, the Virtual Machine checks to see if a previously scoped variable of same name exists and has scope in current location, in the following order:

- system variable (*s.var* or *system.var*)
- local variable (*l.var* or *local.var*)
- database variable (*d.var* or *database.var*)
- global variable (*g.var* or *global.var* or *var*)
- if no variable *var* (with any prefix) exists, then, *var* is created as a global variable

---

**Note:** Use `miva__getvarlist(scope)` to see currently defined variables with the scope.

---

If the variable will be used in an expression, the variable name must start with a letter or an underscore.

## Built-In System Variables

System variables are automatically initialized when a Miva Script program begins execution. There are two types of system variables:

- *Static* system variables  
values are set only when script begins execution.
- *Dynamic* system variables  
values are set each time they are used in an expression.

All of the time-related variables (except for *dyn\_time\_remaining* and *globaltimeout*) have both static and dynamic versions: the dynamic variables start with the *dyn\_* prefix. With the exception of *recno*, all other system variables are static.

### Time Variables

Dynamic	Static	Return value
dyn_time_t	time_t	Number of seconds since 1 Jan. 1970 (numeric)
dyn_tm_hour	tm_hour	Hour in current day (numeric)
dyn_tm_isdst	tm_isdst	Has the value true (1) if daylight time is in effect in this time zone (boolean)
dyn_tm_mday	tm_mday	Day of the month (numeric)
dyn_tm_min	tm_min	Minutes in current hour (numeric)
dyn_stm_mon	stm_mon	Month of the year (string)
dyn_tm_mon	tm_mon	Month of the year (numeric)
dyn_tm_sec	tm_sec	Seconds in current minute (numeric)
dyn_tm_usec	tm_usec	The current microsecond, relative to <i>dyn_tm_sec</i> or <i>tm_sec</i> . (On Windows, this value is updated only in 50-millisecond increments).
dyn_stm_wday	stm_wday	Day of the week (string)
dyn_tm_wday	tm_wday	Day of the week (numeric)
dyn_tm_yday	tm_yday	Day in year (numeric)
dyn_tm_year	tm_year	Year (numeric)
dyn_stm_zone	stm_zone	The time zone (string)
---	globaltimeout	Maximum total number of seconds that this script can execute.
dyn_time_remaining	---	Number of seconds before the current script will time out.

**Note:** The values returned by the *dyn\_tm\_* variables are not zero-padded; for example, if the time is 12:04, *dyn\_tm\_min* returns '4', not '04'. You can use the `padl()` function to perform padding, if required.

### CGI, HTTP, and Other Variables

All available CGI environment variables are automatically converted into static Miva Script system variables upon start-up. All HTTP headers are saved in environment variables and therefore are also converted to Miva Script static variables. The availability of environment variables depends on the server software; the availability of HTTP headers also depends on the browser; therefore, not all variables listed here are guaranteed to be available in all circumstances. For more information on HTTP headers and environment variables, consult a CGI reference and/or your server documentation.

**Note:** HTTP servers convert HTTP headers into environment variables in which '-' is converted to '\_', and to which the prefix 'HTTP' is added.

#### Accessibility Codes

##### Origin:

**S:** generated by Miva Virtual Machine, Miva Mia

**E:** if available, inherited from environment

**H:** if available, inherited from HTTP header (via the environment)

##### Platform:

**C:** accessible with CGI version of Miva Virtual Machine

**N:** accessible with NSAPI version of Miva Virtual Machine

**P:** accessible with Miva Mia

For example, **E: C,N** means that the variable in question is inherited from the environment and is accessible in the CGI and NSAPI versions of Miva Virtual Machine.

Variable	Return Value	Accessibility
apitype	Platform: 'CGI', 'NSAPI', or 'Mia' (Miva Mia)	S: C,N,P
argN	If a list of values is passed to a Miva Script program, <i>argN</i> is the value of the Nth argument on the URL used to call the script. <i>arg1</i> always contains the program file name. The first argument after the file name will be <i>arg2</i> , and so forth.	S: C,N,P
auth_type	Authentication method user by the server	E: C,N
callerid	Each time a cookie-enabled browser accesses a Miva Script document, Miva Script creates a 32-character cookie that is unique to that browser and URL. The cookie lasts for one year after being set. Cookies can be turned off in Miva Virtual Machine; contact your server administrator.	S: C,N,P
content_length	Length of any attached (POST) information	E: C
content_type	Type of data for POST	E: C

## Built-In System Variables

Variable	Return Value	Accessibility
documenturl	Contains URL of the currently running Miva Script program. This URL also contains the character between the program name and the command line arguments ('+' for NSAPI and Miva Mia, and '?' for CGI).	S: C,N,P
gateway_interface	Version of CGI used	E: C
globaltimeout	Maximum total number of seconds that this script can execute.	S: C,N,P
http_accept	Comma-separated list of MIME types ( <i>type/subtype</i> ) that the browser will accept. This list is very incomplete on most browsers.	H: C,N,P
http_accept_charset	Character sets preferred by the browser (other than the default ASCII or ISO Latin-1)	H: C,N,P
http_accept_language	ISO codes for the languages preferred by the browser	H: C,N,P
http_connection	String that browser sends to the server to preserve a TCP connection (also called a "keep-alive" string). Not supported by all browsers and servers.	H: C,N,P
http_cookie	Contents of all the cookies set for the document.	H: C,N,P
http_host	Remote host name (usually same as <b>server_name</b> )	H: C,N,P
http_pragma	Mode client is running under	H: C,P
http_referer	The document that the current document was accessed from.	H: C,P
http_user_agent	Browser name, platform, version, and library	H: C,N,P
mivaversion <b>new in 4.00</b>	Version of the Miva Script language preprocessor (Replaces the previous s.version)	S: C,N,P
miva_defaultlanguage	Retrieves the current default language setting	
miva_language	Retrieves the current language setting	
miva_sslavailable <b>new in 4.00</b>	Whether OpenSSL is available for use by <MvCALL>	S: C,N,P
nargs	If a list of values is passed to a Miva Script program, <i>nargs</i> is the number of arguments on the URL, including the program file name.	S: C,N,P
path_info	Extra path information in the URL (a directory path that occurs immediately after the name of the CGI program in the URL)	E: C,N
path_translated	<i>path_info</i> , translated to a physical location by prepending the server document directory to its value	E: C
process_id	Currently running process number	S: C,N,P
query_string	Information passed after a URL and a "?" via the GET method	E: C

## Built-In System Variables

Variable	Return Value	Accessibility
remote_addr	IP address of remote host	E: C,N,P
remote_host	The domain name of the remote host	E: C,N,P
remote_ident	Remote user name, from servers that support RFC 931 identification	E: C
remote_user	User name associated with protected script, on servers that support user authentication	E: C,N
request_method	GET, POST, or HEAD	E: C,N,P
script_name	Virtual path to CGI script; not mapped locally to actual path	E: C
server_hostname	The name of the server	E: C,N
server_name	Same as server_hostname	E: C,N,P
server_port	Port under which server is running	E: C,P
server_port_secure	Whether the port is secure (boolean)	E: C
server_protocol	Name and revision of protocol used	E: C,N,P
server_software	HTTP server and version number that processed the request	E: C,P
server_url	<b>server_hostname</b> , in URL format	E: C,N
server_version	Version of the server	E: C,N
user_agent	Same as http_user_agent	H: N

Miva Script also allows site administrators to define their own system variables. Check with your administrator to find out whether any have been added to your site.

---

**Note:** Miva does not give scripts direct access to standard input (STDIN).

---

## Functions

- Define functions using <MvFUNCTION> tag.
- Use <MvEVAL> to display the function's returning value.
- Use <MvASSIGN> to return a function's value(s) with no display, and store the function's value.

---

### Passing Variables by Reference

Parameters are variables whose scope is usually the body of the function. But, a parameter may have an optional "VAR" after the variable name, signifying that the parameter would be passed by reference. Any changes to the variable within the function will be reflected in the variable used in calling the function, and any array elements or structure members will be preserved.

For example:

```
<MvFUNCTION NAME="functionname" PARAMETERS="varname1,varname2 VAR,
varname3">
...
</MvFUNCTION>
```

## Built-in Functions

---

### Time Functions

**Note:** These functions cannot be used to process dates earlier than January 1, 1970, or later than January 19, 2038.

---

Function Name (parameters)	Action
ftime ( path ) <b>(new in 4.00)</b>	Returns <i>time_t</i> since a file in the data directory was last modified. See "Time Variables" for <i>time_t</i> definition.
stime( path ) <b>(new in 4.00)</b>	Returns <i>time_t</i> since a file in the script directory was last modified. See "Time Variables" for <i>time_t</i> definition.
time_t_month(time_t, time_zone)	Returns the current month as a number
time_t_year(time_t, time_zone)	Returns the current year
time_t_hour(time_t, time_zone)	Returns current hour (using a 24-hour clock)
time_t_min(time_t, time_zone)	Returns the current minute in the hour
time_t_sec(time_t, time_zone)	Returns the current second in the minute
time_t_dayofmonth(time_t, time_zone)	Returns the current day of the month
time_t_dayofweek(time_t, time_zone)	Returns the current day of the week as a number (Sunday=1)
time_t_dayofyear(time_t, time_zone)	Returns the number of days since the beginning of the year, including today
timezone()	Returns an integer which is the number of hours behind or ahead of GMT ( <i>not</i> accounting for Daylight Time)
mktime_t(year, month, dayofmonth, hours, minutes, seconds, time_zone)	Returns the <i>time_t</i> value for the time specified

## Text String Functions

**Note:** Literal strings or characters used as arguments to these functions must be surrounded by single quotes, '...'. For example: *isalpha('r2d2')*. These functions do not modify their arguments; they return values based on those arguments.

### Boolean-valued String Functions

These functions all start with *is* and return a true (1) or false (0) value depending on the composition of the string. Each of these functions is based on the C language function of the same name, but is applied to the whole string: *isdigit(string)* will return true if every character in the string is a digit or null, and false otherwise.

**Note:** For all functions except *isdigit()* and *isxdigit()*, the set of characters understood to be alphabetic is inherited from the *setlocale()* setting used on the system running the Miva Virtual Machine. For this reason, these functions are not guaranteed to return the same results on all machines.

Function Name (parameters)	Action
<i>isalnum(string)</i>	Returns true (1) if all characters in <i>string</i> are either alphabetic or digits, and false (0) otherwise.
<i>isalpha(string)</i>	Returns true (1) if all characters in <i>string</i> are alphabetic and false (0) otherwise.
<i>isascii(string)</i>	Returns true (1) if all characters in <i>string</i> are ASCII characters (those with decimal value between 0 and 127), and false (0) otherwise.
<i>iscntrl(string)</i>	Returns true (1) if all characters in <i>string</i> are control characters (those with decimal value between 0 and 31, or 127), and false (0) otherwise.
<i>isdigit(string)</i>	Returns true (1) if all characters in <i>string</i> are digits in the range 0-9, and false (0) otherwise.
<i>isgraph(string)</i>	Returns true (1) if all characters in <i>string</i> are graphic characters (those with decimal value between 33 and 127), and false (0) otherwise.
<i>islower(string)</i>	Returns true (1) if all characters in <i>string</i> are lowercase letters, and false (0) otherwise.
<i>isprint(string)</i>	Returns true (1) if all characters in <i>string</i> are printable characters (same as graphic characters, with the addition of the space character), and false (0) otherwise.
<i>ispunct(string)</i>	Returns true (1) if all characters in <i>string</i> are punctuation characters (non-alphanumeric graphics characters), and false (0) otherwise.
<i>isspace(string)</i>	Returns true (1) if all characters in <i>string</i> are whitespace (space, tab, vertical tab, newline, form feed) characters, and false (0) otherwise.
<i>isupper(string)</i>	Returns true (1) if all characters in <i>string</i> are uppercase letters, and false (0) otherwise.
<i>isxdigit(string)</i>	Returns true (1) if all characters in <i>string</i> are hexadecimal digits (a-f, A-F, 0-9), and false (0) otherwise.

Other String Functions

Function Name (parameters)	Action
asciichar (number)	Returns the character corresponding to <i>number</i> ( <i>number</i> must be less than 255).
asciivalue (character)	Returns the ASCII numeric value for <i>character</i> ( <i>character</i> must be a single character).
decodeattribute (string)	Returns a copy of <i>string</i> (which is usually a URL) converted from URL-encoded format to ordinary text. This function is the opposite of encodeattribute.
decodeentities (string)	Returns a copy of <i>string</i> in which all HTML entities have been converted to their plain text equivalents (for example, '&lt;' is converted to '<'). This function is the opposite of encodeentities.
encodeattribute (string)	Returns a copy of <i>string</i> (which is usually a URL) in URL-encoded format. Special characters such as space, tilde (~), and the plus sign are converted to hexadecimal %nn format. This function is the opposite of decodeattribute.
encodeentities (string)	Returns a copy of <i>string</i> in which all characters have been converted to their HTML entity equivalents, where applicable (for example, '<' is converted to '&lt;'). This function is the opposite of decodeentities.
gettoken(string, separators, n)	Tokenizes <i>string</i> , using any of the characters in <i>separators</i> as token separators, and returns the <i>n</i> th token. A null string is returned if there is no <i>n</i> th token.
glosub_array( string, search, replace ) <b>(new in 4.00)</b>	Works like glosub(), but <i>search</i> and <i>replace</i> are arrays that are iterated through, each value in the <i>search</i> array found in the string is replaced by the corresponding <i>replace</i> array value.
glosub(string, search, replace)	Global substitution; returns a copy of <i>string</i> in which all instances of string <i>search</i> have been replaced by string <i>replace</i> . (Note: to represent a backslash (\) in <i>replace</i> , use '\\').
len(string)	Returns the number of characters in <i>string</i> .
ltrim(string)	Returns a copy of <i>string</i> with all space characters removed from the left end.
padl(string, length, padcharacter)	Returns a string <i>length</i> characters long, consisting of <i>string</i> padded on the left with as many instances of <i>padcharacter</i> as are needed to make up the full length.
padr( string, length, padcharacter)	Returns a string <i>length</i> characters long, consisting of <i>string</i> padded on the right with as many instances of <i>padcharacter</i> as are needed to make up the full length.
rtrim(string)	Returns a copy of <i>string</i> with all space characters removed from the right end.
substring(string, start, length)	Returns the substring of <i>string</i> , beginning at position <i>start</i> , <i>length</i> characters long.

Function Name (parameters)	Action
tokenize( string, replacements ) <b>(new in 4.00)</b>  <b>Example:</b> <pre>&lt;MvASSIGN NAME = "l.replacements" INDEX = "{1}" MEMBER = "token" VALUE = "token1"&gt; &lt;MvASSIGN NAME = "l.replacements" INDEX = "{1}" MEMBER = "value" VALUE = "value"&gt; &lt;MvASSIGN NAME = "l.replacements" INDEX = "{1}" MEMBER = "token" VALUE = "token2"&gt; &lt;MvASSIGN NAME = "l.replacements" INDEX = "{1}" MEMBER = "value" VALUE = "works"&gt; &lt;MVASSIGN NAME = "l.tokens" VALUE = "{ tokenize( 'This is a %token1% that %token2%.' , l.replacements ) }"&gt; &lt;MvEVAL EXPR="{ l.tokens }"&gt;</pre>	Returns the string, concatenated with the value of each token contained in replacements. Replacements is an array of structures, each with a token and a value.
tolower(string)	Returns a copy of <i>string</i> in lower case.
toupper(string)	Returns a copy of <i>string</i> in upper case.
trim( string ) <b>(new in 4.00)</b>	Returns the value of <i>string</i> with leading and trailing spaces removed

---

**Note:** The *asciichar(n)* function puts out a single byte whose value is *n*. It is up to the client application (such as a browser) to render this value as a character. Almost all HTML browsers support the ISO Latin-1 character encoding, and will display *asciichar(n)* as the character whose ISO Latin-1 (decimal) encoding is *n*. The rendering of non-printable characters, and values of *asciichar(n)* where *n* is not associated with a character in ISO Latin-1, is undefined. In particular, the rendering of *asciichar(n)* where *n* is in the decimal range 129-160, is undefined, though many browsers will display the corresponding ANSI character. The rendering of the output of *asciichar(n)* by an arbitrary application follows the character encoding used by that application, and is in general platform-specific.

---

## Numerical Functions

The arguments of the trigonometric functions *sin*, *cos*, *tan*, *sinh*, *cosh*, and *tanh* should be expressed in *radians*; the results of *asin*, *acos*, *atan*, and *atan2* are expressed in radians. Function *arguments* in radians can be given in the range 0 to 2pi, or -pi to pi.

Function Name (parameters)	Action
acos(number)	Returns the arccosine of <i>number</i> .
asin(number)	Returns the arcsine of <i>number</i> .
atan(number)	Returns the arctangent of <i>number</i> .
atan2(y,x)	Returns the arctangent of <i>y/x</i> . This is similar to <i>atan()</i> , but the signs of <i>y</i> and <i>x</i> are taken into account when computing the quadrant of the result.
ceil(number)	Returns the smallest integer greater than or equal to <i>number</i>
cos(number)	Returns the cosine of <i>number</i> .
cosh(number)	Returns the hyperbolic cosine of <i>number</i> .
exp(number)	Returns the constant <i>e</i> (approximately 2.71828) raised to the power <i>number</i> .
floor(number)	Returns the largest integer less than or equal to <i>number</i>
fmod(number1,number2)	Returns the remainder of <i>number1/number2</i> ; <i>number1</i> , <i>number2</i> , and the result are all floating point numbers.
int(number)	Returns integer portion of <i>number</i> (removes the decimal and any digits to the right of it)
log(number)	Returns the natural logarithm (base <i>e</i> , approximately 2.71828) of <i>number</i> .
log10(number)	Returns the base 10 logarithm of <i>number</i>
power(number, power)	Raises <i>number</i> to a a power (for example, <i>power(12,2)=144</i> )
random(maximum)	Returns a random number less than or equal to <i>maximum</i>
rnd(number, number_of_places)	Works like the ROUND operator, rounding <i>number</i> up or down to <i>number_of_places</i> after the decimal
sin(number)	Returns the sine of <i>number</i> .
sinh(number)	Returns the hyperbolic sine of <i>number</i> .
sqrt(number)	Returns the square root of <i>number</i> .
tan(number)	Returns the tangent of <i>number</i> .
tanh(number)	Returns the hyperbolic tangent of <i>number</i> .

## File System Functions

- These functions operate on files in the Miva data directory (for *f*-functions) and the Miva script directory (for *s*-functions)
- *fs*- and *sf* functions operate on both directories. (The first character, *f* or *s*, is the source, second is the destination.)
- *Literal* filenames and paths must be surrounded by single quotes.  
... <MvIF EXPR="{fexists('mondo.dat')}">...
- Forward slash (on Windows and Unix) separates folders/directories.
- Except as indicated, all file system functions return a value of '1' (true) if they succeed, and '0' (false) if they fail.

Function Name(parameters)	Action
fchmod(source_path, mode_number) schmod(source_path, mode_number)	<i>fchmod()</i> works like UNIX <i>chmod</i> and changes permissions to <i>mode_number</i> on the file named by <i>source_path</i> in the data directory. <i>schmod()</i> does the same in the script directory. The mode number must be in the format <i>nnnn</i> (decimal) or <i>Onnnn</i> (octal). <b>New in 4.01:</b> Literal Octal requires single quotes. Example: <MvASSIGN NAME = "varname" VALUE = "{ fchmod( source_path, '0755' ) }">
fcopy(source_path, destination_path) scopy(source_path, destination_path)	<i>fcopy()</i> work like UNIX <i>cp</i> to copy the file named by <i>source_path</i> to the file named by <i>destination_path</i> , both within the data directory; <i>scopy()</i> does the same in the script directory.
fscopy(source_path, destination_path) sfcopy(source_path, destination_path)	<i>fscopy()</i> works like UNIX <i>cp</i> to copy the file named by <i>source_path</i> in the data directory to the file named by <i>destination_path</i> in the script directory; <i>sfcopy()</i> does the same thing in the reverse direction.
fdelete(path), sdelete(path)	<i>fdelete()</i> works like UNIX <i>rm</i> to delete the file named by <i>path</i> in the data directory. <i>sdelete()</i> deletes the file or directory named by <i>path</i> in the script directory.
fexists(path), sexists(path)	<i>fexists()</i> returns a boolean true (1) or false (0) testing if the file named by <i>path</i> exists in the data directory; <i>sexists()</i> does the same for the script directory.
fmkdir(path), smkdir(path)	<i>fmkdir()</i> works like UNIX <i>mkdir</i> to create a directory specified by <i>path</i> in the data directory; <i>smkdir()</i> does the same in the script directory.
fmode(path), smode(path)	Returns the permissions mode of the file named by <i>path</i> in the data directory (returns -1 if the file does not exist). <i>smode()</i> does the same in the script directory.
frename(source_path, destination_path), srename(source, destination)	<i>frename()</i> works like UNIX <i>mv</i> to rename the file named by <i>source_path</i> to be the file named by <i>destination_path</i> , both within the data directory. <i>srename()</i> does the same in the script directory.

Function Name(parameters)	Action
<code>fsrename(source_path, destination_path), sfrename(source_path, destination_path)</code>	<i>fsrename()</i> works like UNIX <i>mv</i> to rename the file named by <i>source_path</i> in the data directory to be the file named by <i>destination_path</i> in the script directory. <i>sfrename()</i> does the same in the opposite direction.
<code>fsize(path), ssize(path)</code>	<i>fsize()</i> returns the number of bytes in the file named by <i>path</i> in the data directory (returns -1 if the file does not exist). <i>ssize()</i> does the same in the script directory.
<code>fsymlink(file, lnk)</code> (Unix only)	Creates a symbolic link to the file in the data directory. file - path name of the file relative to the Miva data directory. lnk - path name of the link relative to the Miva data directory.
<code>ssymlink(file, lnk)</code> (Unix only)	Creates a symbolic link to the file in the scripts directory. file - path name of the file relative to the Miva scripts directory. lnk - path name of the link relative to the Miva scripts directory

---

## Language Functions

Function Name (parameters)	Action
<code>miva_setlanguage (language)</code>	sets the current language. <i>Language</i> has two parts: language-country, such as en-US for English-United States.
<code>miva_setdefaultlanguage (language)</code>	sets the current default language.

## System Functions

Function Name (parameters)	Action
makesessionid( )	Returns a 128-bit unique ID.
miva_array_collapse(array) <b>(new in 4.00)</b>	Returns a copy of the array with the indices sequential from one to the number of elements in the source array. (So, array[1], array[50], array[100] would be resultarray[1], resultarray[2], resultarray[3].)
miva_array_elements	This function returns the number of elements in the array that were actually used.
miva_array_max	This function will return the maximum array element used. Example: <MvASSIGN NAME="l.array" INDEX="1" VALUE="n"> <MvASSIGN NAME="l.array" INDEX="100" VALUE="m"> <MvASSIGN NAME="max" value="{miva_array_max(l.array)}"> MIVA_ARRAY_MAX returns the value of 100.
miva_array_serialize miva_array_deserialize	miva_array_serialize returns a string representation of the array and any subarrays. miva_array_deserialize is passed a string in the format returned from miva_array_serialize, and returns an aggregate repopulated to match the original aggregate.
miva_getvarlist (scope)	Returns a comma-separated list of the names of all currently defined variables with the given scope. <i>scope</i> must be a literal string: 'l', 'local', 'g', 'global', 's', 'system'. ( <i>Note: For xBase3 databases, use &lt;MvREVEALSTRUCTURE&gt;.</i> )
miva_output_flush( ) <b>new in 4.00</b>	Writes the HTTP headers and any other output to the browser. Second and subsequent calls will write the output, but not rewrite the headers.
miva_output_header( name, value) <b>new in 4.00</b>	Sets an HTTP header name-value pair.
miva_variable_value (varname)	Takes a variable name and returns the value of the variable to which the value of <i>varname</i> is referring.

**Encryption Functions**

Function Name (parameters)	Action
rsa_generate_keypair(pubkey_file, privkey_file, bits, e, passphrase)	Generates an RSA keypair, saving the public key in "pubkey_file", the private key in "privkey_file", and encrypting the private key with "passphrase". Returns 1 on success, 0 on error.
rsa_load_publickey(pubkey_file, rsa VAR)	Load an RSA public key from a PKCS#1 file specified by "pubkey_file". Returns 1 on success, 0 on error.
rsa_load_privatekey(privkey_file, rsa VAR, passphrase)	Load an encrypted RSA private key from a PKCS#8 file specified by "privkey_file", and decrypt it using "passphrase". Returns 1 on success, 0 on error.
rsa_public_encrypt(rsa, plaintext, encrypted VAR)	Encrypt the data in "plaintext", storing the result in "encrypted", using the public key portion of the RSA structure specified by "rsa". Returns 1 on success, 0 on error.
rsa_public_decrypt(rsa, encrypted, plaintext VAR)	Decrypt the data in "encrypted", storing the result in "plaintext", using the public key portion of the RSA structure specified by "rsa". Returns 1 on success, 0 on error.
rsa_private_encrypt(rsa, plaintext, encrypted VAR)	Encrypts the data in "plaintext", storing the result in "encrypted", using the private key portion of the RSA structure specified by "rsa". Returns 1 on success, 0 on error.
rsa_private_decrypt(rsa, encrypted, plaintext VAR)	Decrypts the data in "encrypted", storing the result in "plaintext", using the private key portion of the RSA structure specified by "rsa". Returns 1 on success, 0 on error.
rsa_free(rsa VAR)	Frees the RSA indicated by "rsa". Returns 1 on success, 0 on error.
crypto_rand_bytes(n)	Generates "n" random bytes, and returns them.
crypto_base64_encode(data)	Base-64 encodes "data", returning the encoded result.
crypto_base64_decode(data)	Base-64 decodes "data", returning the decoded result.
crypto_md5(data)	Calculates the MD5 hash of "data", returning the result.
bf_encrypt(key, plaintext, encrypted VAR)	Blowfish encrypts "plaintext" using the key "key" in ECB mode, storing the result in "encrypted". Returns 1 on success, 0 on error.
bf_decrypt(key, encrypted, plaintext VAR)	Blowfish decrypts "encrypted" using the key "key" in ECB mode, storing the result in "plaintext". Returns 1 on success, 0 on error.

## Assignment and Evaluation Tags

### <MvASSIGN>

#### Attributes

NAME="{expression} | literal"  
 VALUE="{expression} | literal"  
 MEMBER="member:submember"  
 INDEX="{expression} | literal"

#### Comments

- Empty tag.
- **NOTE:** If a value is assigned to an array without specifying an index, the array will be destroyed

#### Syntax:

```
<MvASSIGN NAME="var" VALUE="{expression} | literal">
<MvASSIGN NAME="array[3]" value="{xxx}"
<MvASSIGN NAME="l.structure" MEMBER="member:submember" VALUE="2">
<MvASSIGN NAME="varname:membername" INDEX="3" VALUE="{xxx}">
```

### <MvASSIGNARRAY>

#### Attributes

NAME="{expression} | literal"

#### Comments

- Closed tag.
- Defines a multi-dimension array
- Only <MvDIMENSION> and <MvMEMBER> tags allowed within <MvASSIGNARRAY> block.

#### Syntax:

```
<MvASSIGNARRAY NAME="l.array"
VALUE="123">
<MvDIMENSION INDEX="1">
<MvDIMENSION INDEX="2">
<MvDIMENSION INDEX="7">
</MvASSIGNARRAY>
(Accessed in an expression as "l.array[1][2][7]" )
```

### <MvDIMENSION>

#### Attributes

INDEX="{expression} | literal"

#### Comments

- Empty tag.
- Defines Multi Dimension Array Index within <MvASSIGNARRAY> block.
- MvDIMENSION and MvMEMBER tags may be intermixed
- **NOTE:** Must be within the <MvASSIGNARRAY> block.

#### Syntax:

```
<MvASSIGNARRAY NAME="varname" VALUE="vvv">
<MvDIMENSION INDEX="4">
<MvDIMENSION INDEX="5">
<MvDIMENSION INDEX="6">
</MvASSIGNARRAY>
```

### <MvMEMBER>

#### Attributes

NAME="{expression} | literal"

#### Syntax:

```
<MvASSIGNARRAY NAME="l.array" VALUE="123">
  <MvMEMBER NAME="one">
  <MvMEMBER NAME="two">
  <MvMEMBER NAME="seven">
</MvASSIGNARRAY>
```

#### Comments

- Empty tag.
- Defines a Multi-dimension Array Index within <MvASSIGNARRAY> block.
- Additional member attributes can be included in the MEMBER attribute of MvASSIGN.
- MvDIMENSION and MvMEMBER tags may be intermixed.

---

### <MvEVAL>

#### Attributes

EXPR="{expression}"

EXPRESSION="{expression}"

#### Syntax:

```
<MvEVAL EXPR = "{var1 + var2}">
<MvEVAL EXPR="{ 'The answer: ' $ func1(42)}">
```

#### Comments

- Empty tag.
- Displays expression results to browser.
- 

## Configuration and Function Tags

---

### <MIVA>

#### Attributes

INTERPRET="html\_expressions

STANDARDOUTPUTLEVEL="html, text, compresswhitespace"

ERROROUTPUTLEVEL="syntax,expression,runtime"

ERRORMESSAGE="text\_of\_error\_message"

MvTAGNAME\_ERROR="fatal/nonfatal,display/nodisplay">

#### Syntax:

```
<MIVA STANDARDOUTPUTLEVEL = "text, html, compresswhitespace">
```

#### Comments

- Empty tag.
- INTERPRET has no behavior for compiled Miva Script.
- STANDARDOUTPUTLEVEL specifies what displays in browser
- Runtime value for ERROROUTPUTLEVEL is the only useful flag for Compiled Miva Script. (Syntax and Expression errors are flagged by the compiler.)
- *MvTAGNAME* is a Miva tag
- *fatal* causes runtime error for tag to terminate script
- *display* causes runtime error for named tag to display.

---

### <MvCOMMENT>

#### Attributes

(none)

#### Syntax:

```
<MvCOMMENT>This is a comment to explain my code.</MvCOMMENT>
```

#### Comments

- Closed tag.

### <MvFUNCTIONRETURN>

#### Attributes

VALUE="{expression} | literal"

#### Syntax:

```
<MvFUNCRETURN VALUE="{expression}">
```

#### Comments

- Empty tag.
  - Returns a value back to the tag that called the function.
- Equivalent name: <MvFUNCRETURN>

### <MvFUNCTION>

#### Attributes

NAME="{expression} | literal"  
 PARAMETERS="var1,var2 VAR,var3..."  
 STANDARDOUTPUTLEVEL="html,text,  
 compresswhitespace"  
 ERROROUTPUTLEVEL="syntax,expression,runtime"

#### Syntax:

```
<MvFUNCTION NAME="functionname"  

  PARAMETERS="var1,var2 VAR, var3">  

  ...  

</MvFUNCTION>
```

#### Comments

- Empty tag.
- NAME is required.
- Parameters are optional.
- Runtime value for ERROROUTPUTLEVEL is the only useful flag for Compiled Miva Script. (Syntax and Expression errors are flagged by the compiler.)
- Parameter may be passed by reference entering VAR after variable. (changes to variable within this function will be reflected in the variable used in calling this function.)

### <MvHIDE>

#### Attributes

FIELDS="var1,var2,var3"

#### Syntax:

```
<MvHIDE FIELDS="v1,v2,v3">
```

#### Comments

- Empty tag.
- Use only inside FORM
- Passes hidden fields on the fly, when a form is submitted.
- Variables and values passed as name/value pairs (specified by the <FORM>'s ACTION attribute) or in the standard input if METHOD is POST.
- Converts into one or more hidden <INPUT> tags, corresponding to each variable specified by FIELDS.

## Localization Tags

### <MvLOCALIZED>

#### Attributes

NAME="var" (*optional*)  
 ID="id-001"  
 STANDARDOUTPUTLEVEL="html,text,  
 compresswhitespace" (*or null*)

#### Comments

- Closed tag.
- Starts and ends Localization block.
- If NAME specified, output is stored in named variable, instead of output to the browser.

**NOTE:** Built-in language functions set the current and default language. See ["Language Functions" on page 17.](#)

---

**<MvLOCALIZED-TOKEN>****Attributes**

NAME = "token\_name"  
 VALUE="{EXPR} | literal"

**Syntax:**

```
<MvLOCALIZED-TOKEN NAME="token" VALUE="xxx" >
```

**Comments**

- Empty tag.
- Must be inside <MvLOCALIZED> block.
- Must precede <MvLocalized-Text> block(s) that use token.

---

**<MvLOCALIZED-TEXT>****Attributes**

LANGUAGE="language\_id"

**Syntax:**

```
<MvLOCALIZED-TEXT LANGUAGE = "en-US" >
<P>Note the new rates of 5%%.</P>
</MvLOCALIZED-TEXT>
```

**Comments**

- Closed tag.
- The text for the language specified by the LANGUAGE attribute.
- HTML allowed within block.
- Miva tags not allowed within block.

**Note:** The percent character, %, is used to identify a token. Therefore, to use it within the localized text, enter two in a row, as shown in the sample above.

---

**Database Tags**

---

**<MvADD>****Attributes**

NAME = "database alias" (*optional*)  
 View = "viewname" (*optional*)

**Syntax:**

```
<MvADD> NAME = "Alias"
```

**Comments**

- Empty tag.
- Adds record of defined fields to database.
- If name omitted, record is added to primary database.

---

**<MvCLOSE>****Attributes**

NAME="database alias" (*optional*)

**Syntax:**

```
<MvCLOSE NAME = "Products" >
```

**Comments**

- Empty tag.
- Closes the database alias. If name omitted, primary database alias is closed.

**NOTE:** Database might continue to be open under a different alias.

---

## <MvCLOSEVIEW>

### Attributes

NAME = "database alias"  
VIEW = "viewname"

### Syntax:

```
<MvCLOSEVIEW NAME="db_alias"
VIEW="viewname" >
```

### Comments

- Empty tag.
- Closes the specified SQL view.

---

## <MvCREATE>

### Attributes

NAME = "database alias"  
DATABASE = "db\_file | {exp}"  
FIELDS = "fieldname1 CHAR(max\_chars),  
    fieldname2 NUMBER (digits\_before.digits\_after),  
    fieldname3 DATE,  
    fieldname4 BOOL,  
    fieldname5 MEMO "  
TYPE="xbase3" (*optional, defaults to xBase3*)

### Comments

- Empty tag.
  - Creates new database file, with specified name and fields.
- NOTE:** An alias becomes the primary alias automatically when it is created or opened (subsequent create or open operations override this).

---

## <MvDELETE>

### Attributes

NAME = "database alias"  
VIEW = "viewname" (*optional for xBase3*)

### Syntax:

```
<MvDELETE NAME = "Categories">
```

### Comments

- Empty tag.
- Marks record for deletion.

See [<MvPACK>](#) and "[<MvUNDELETE>](#)"

---

## <MvFILTER>

### Attributes

NAME = "database alias"  
FILTER\_TYPE="expression | variable" (*optional, defaults to expression*)  
FILTER="{expression}"  
NAME = "database alias"

### Comments

- Empty tag.
- Records not matching FILTER condition are invisible to navigation tags
- When FILTER\_TYPE is variable, the FILTER attribute must be an expression that resolves to a variable name.
- Omit FILTER attribute to remove a filter.

**NOTE:** The variables *totrec* and *recno* continue to refer to the complete, unfiltered database

---

**<MvFIND>****Attributes**

NAME="database alias"  
 VALUE="{expression} | literal"  
 EXACT (*optional*)  
 VIEW = viewname (*optional for xBase3*)

**Comments**

- Empty tag.
- Performs case sensitive search.
- Points to first matching record.
- If NAME is omitted, uses primary database's index.

---

**<MvGO>****Attributes**

NAME="database alias"  
 ROW="row\_number | top | bottom"  
 VIEW="odbc\_db\_view" (*not required for internal xBase3 databases*)

**Comments**

- Empty tag.
- Row\_number refers to physical order.
- Top and bottom refer to indexed order.

**Syntax:**

```
<MvGO NAME = "Categories" ROW = "top">
```

---

**<MvOPEN>****Attributes**

NAME="database alias"  
 DATABASE="{expression} | literal"  
 TYPE="xbase3 | odbc | *user-defined*" (*optional for xbase3*)  
 INDEXES="index1.mvx,index2.mvx,..." (*optional*)  
 username (*non-xBase3*)  
 password (*non-xBase3*)

**Comments**

- Empty tag.
- The same database file can be open more than once simultaneously with different aliases.

**NOTE:** An alias becomes the primary alias automatically when it is created or opened (subsequent create or open operations override this).

---

**<MvOPENVIEW>****Attributes**

NAME="database alias"  
 VIEW="view name"  
 QUERY="sqlquery"  
 Fields="varlist"

**Syntax:**

```
<MvOPENVIEW NAME = "Example"  

VIEW = "Products"  

QUERY = "SELECT * FROM prod_table"  

  where id = :A" (Oracle)  

  where id = ?" (ODBC)>
```

**Comments**

- Empty tag.
- <MvOPENVIEW> opens a view based on the results of the sqlquery.
- <MvGO> and <MvSKIP> can be used with views. Other tags that allow the VIEW attribute:
  - <MvPRIMARY>
  - <MvFIND>
  - <MvSETINDEX>
  - <MvUPDATE>
  - <MvDELETE>
  - <MvUNDELETE>
  - <MvADD>
  - <MvREVEALSTRUCTURE>

### <MvPACK>

#### Attributes

NAME="database alias"

#### Comments

- Empty tag.
- Permanently removes all records that were marked for deletion (with <MvDELETE>).
- Physical record numbers are reset.

**NOTE:** All open index files are automatically updated after <MvPACK >is executed. (Use <MvREINDEX for closed indexes.)

---

### <MvPRIMARY>

#### Attributes

NAME="database alias"

INDEX = "index1.mvx"

#### Comments

- Empty tag.
  - Explicitly names the primary index.
- 

### <MvQUERY>

#### Attributes

NAME="database alias" (*optional*)

QUERY="SQL query"

FIELDS = "varlist"

#### Syntax:

```
<MvQUERY NAME="my_db"
QUERY = "SELECT * FROM prod_table"
  where id = :A"> (Oracle)
  where id = ?" (ODBC)>
```

#### Comments

- Empty tag.
  - Must have an opened view for queries to return results.
  - If name omitted, applies to primary database.
- 

### <MvREVEALSTRUCTURE>

#### Attributes

NAME="database alias"

DATABASE="path/filename.dbf"

VIEW= "viewname" (*optional for xBase3*)

VARIABLE="varname"

#### Comments

- Creates new database, which is not open.
- Each record contains:
  - field\_name:
  - field\_type
  - field\_len
  - field\_dec

If variable is specified, variable is set to array of structures, with array index = field # and member names as above (listed in "Each record contains")

### <MvSKIP>

#### Attributes

NAME="database alias"

ROWS="number"

VIEW="odbc\_db\_view" (*optional for xBase3*)

#### Syntax:

#### Comments

- Empty tag.
- Moves the record pointer a specific number of records forward or backward relative to its current position.
- If name omitted, applies to primary database alias.
- ROWS defaults to one.

```
<MvIF EXPR = "{ NOT Categories.d.EOF }"> <MvSKIP NAME = "Categories" ROWS = "1">
```

---

### <MvUNDELETE>

#### Attributes

NAME="database alias"

VIEW="viewname" (*optional for xBase3*)

#### Comments

- Empty tag.
- Removes mark for deletion set by <MvDELETE> from the current record in the named database alias.

---

### <MvUPDATE>

#### Attributes

NAME="database alias"

VIEW="viewname" (*optional for xBase3*)

#### Comments

- Empty tag.
- Updates current record with the contents of variables corresponding to the database fields.
- If name omitted, applies to the primary database.
- Locks the record that is being updated.

## Database Index Tags

---

### <MvMAKEINDEX>

#### Attributes

NAME="database alias"  
INDEXFILE="filename.mvx"  
EXPR\_TYPE="{expression} | variable"  
EXPR="{key\_expr} | variablename"  
FLAGS="[ascending | descending],  
[unique | nunique], [string]"

#### Comments

- Empty tag.
- Creates index files for named database alias.
- If name omitted, applies to the primary database.
- Ordered by value of EXPR and FLAGS.
- By default, becomes the Primary Index for the named database alias.

**NOTE:** The key expression may not exceed 500 characters in length, and may not evaluate to more than 100 characters.

---

### <MvREINDEX>

#### Attributes

NAME="database alias" (optional)

#### Comments

- Empty tag.
  - Recreates all open index files for named database alias.
  - If name omitted, primary database index files are reindexed.
- 

### <MvSETINDEX>

#### Attributes

NAME="database alias"  
INDEXES="index1.mvx, index2.mvx"

#### Comments

- Empty tag.
- The specified index file(s) become the index file(s) for the named database alias.
- First index file listed becomes the primary index for named database.
- If name omitted, applies to primary database.

## Database Implicit Fields

Remember that the physical record number of a record may change if the database is packed (that is, some records are physically deleted using <MvPACK>).

Function Name (parameters)	Action
<p><b>deleted</b></p> <p>Works with: xbase3</p> <p>Does NOT Work with: ODBC, Oracle</p>	<p>If the current record has been marked for deletion, the boolean variable <code>db_alias.d.deleted</code> is set to true (deleted or <code>d.deleted</code> can be used when referring to the primary database).</p> <p>Example: <code>db_alias.d.deleted</code></p>
<p><b>eof</b></p>	<p>End of File. The boolean variable <code>db_alias.d.eof</code> is true if the end of the database with alias <code>db_alias</code> has been reached.</p> <p>Example:  <pre>&lt;MvWHILE EXPR="{NOT worker_db.d.eof}" &lt;MvEVAL EXPR="{worker_db.d.employee}" ...</pre> </p>
<p><b>recno</b></p> <p>Works with: ODBC, Oracle, xBase3</p>	<p>Current physical record number in a database or query.</p> <p>Example: <code>db_alias.d.recno</code></p> <p><b>Note:</b> &lt;MvIMPORT&gt; uses <code>recno</code></p>
<p><b>totrec</b></p> <p>Works with: xbase3</p> <p>Does NOT Work with: ODBC, Oracle</p>	<p>Total number of records in the database.</p> <p>Example: <code>db_alias.d.totrec</code></p>

---

## Email Tags

---

### <MvPOP>

#### Attributes

MAILHOST="host"  
 LOGIN="login\_name"  
 PASSWORD="password"  
 DIRECTORY="directory"

#### Comments

- Closed tag.
- Miva Script documents can receive email by becoming POP3 clients.
- Stores received data in special variables:
  - messagesubject
  - messagedate
  - messagesender
  - messagereplyto
  - messagebody

---

### <MvPOPDELETE>

#### Attributes

(none)

#### Comments

- Empty tag.
- Deletes the current email message on the server.
- Must be executed inside the <MvPOP> loop. The message will still be processed by the application in this iteration of the <MvPOP> loop.

**Note:** Since some servers do not remove their temp files, make sure you delete the temp files, where present.

---

### <MvPOPSTOP>

#### Attributes

(none)

#### Comments

- Empty tag.
- Terminates the current <MvPOP>.
- Must be executed inside the <MvPOP> loop.

---

### <MvSMTP>

#### Attributes

TO= "to\_address1,to\_address2,..."  
 SUBJECT="expression" (*optional*)  
 CC="cc\_address1,cc\_address2,..." (*optional*)  
 MAILHOST="mailhost"  
 FROM="from\_address">  
 [optional headers]  
 [blank line]  
 ... message body (text/tags)...

#### Comments

- Closed tag.
- SMTP is used to send mail from any valid mail server
- Optional headers are, Reply-To, Return-Path, and Sender.

---

## External File Tags

---

### <MvDO>

**Attributes**

FILE="filename" (*.mvc only*)  
 NAME="var" (*optional*)  
 VALUE="{func(args)}" (*optional*)

**NOTE:** A .txt or .html file may be renamed to .mvc, then compiled. (Remember to rename the file name in the originating script file.) Also see: "[<MvCALL>](#)".

**Comments**

- Empty tag.
- <MvDO>tags can be nested.
- If NAME and VALUE specified, the named function is the only code in the external file that is executed.
- If NAME and VALUE are omitted, everything in the external file is executed, and the results of all <MvASSIGN> tags are available to subsequent code in the current file. However, any function definitions in the external file are not available after the <MvDO> has been processed.

---

### <MvEXPORT>

**Attributes**

FILE="filename"  
 FIELDS="var1,var2,var3,..."  
 DELIMITER="chars">

**Comments**

- Empty tag.
- Writes a single line of data to an external output file, at the end of the output file, on a new line.
- Does not automatically loop. See "[<MvWHILE>](#)".

---

### <MvIMPORT>

**Attributes**

FILE = "filename"  
 FIELDS = "var1,var2,var3,..."  
 DELIMITER = "chars"  
 FILTER\_TYPE = "{expression} | variable" (*optional*)  
 FILTER = ""{expression} | variablename" (*optional*)

**Comments**

- Closed tag.
- Loops once through entire file, stopping on eof, <MvIMPORT>, or <MvIMPORTSTOP>.
- The variable, *recno*, is automatically created.
- <MvIMPORT> may be nested.

---

### <MvIMPORTSTOP>

**Attributes**

(none)

**Comments**

- Empty tag.
- Stops <MvIMPORT>.
- Jumps to code that follows </IMPORT> tag.

---

### <MvLOCKFILE>

**Attributes**

FILE="filename"

**Comments**

- Empty tag.
- Indicates to other processes that the current process has requested an exclusive lock on FILE.
- Multiple lock requests are queued.
- <MvLOCKFILE> tags can be nested.

---

## Flow Control Tags

---

### <MvELSE>

**Attributes**

(none)

**Comments**

- Empty tag.
- Alternative Script branch.
- Preceded by <MvIF>.

---

### <MvELSEIF>

**Attributes**

EXPR="{expression}"

**New in 4.00**

**Comments**

- Empty tag.
- Use in place of nested <MvIF> tags.

---

### <MvEXIT>

**Attributes**

(none)

**Comments**

- Empty tag.
- Causes the script in which the exit tag resides to terminate.

**NOTE:** An <MvEXIT> within a FUNCTION in a script called with <MvDO> also causes the script containing the <MvDO> to terminate. Therefore, to exit from a script called by <MvDO>, without exiting from the calling script, use <MvFUNCRETURN>.

---

### <MvIF>

**Attributes**

EXPR="{conditional expression}"

**Syntax:**

```
<MvIF EXPR="{file.exists('file.dat')}">
```

**Comments**

- Closed tag.
- Processes the code between the <MvIF> and </MvIF> tags if and only if condition is true.

### <MvWHILE>

#### Attributes

EXPR="{expression}"

#### Comments

- Closed tag.
- Repeats code until condition is false, or, until <MvWHILESTOP> is encountered.

### <MvWHILESTOP>

#### Attributes

(none)

#### Comments

- Empty tag.
- Terminates MvWHILE loop.

## HTTP and Commerce Server Tags

---

### <MvCALL>

#### Attributes

ACTION = "URL"

METHOD = "POST|GET|XML|RAW"

CONTENT-TYPE

FIELDS = "var1,var2,var3,..."

FILES = "fvar1,fvar2,fvar3,...">

**(Methods "XML" and "RAW" new in 4.00)**

#### Comments

- Closed tag.
- HTTP protocol emulates a browser and contacts a remote host.
- When Method = XML, POST content type will be "text/xml." FIELDS contain variables whose values pass in the header.
- When Method = POST, the FIELDS variables' values are sent to the URL.
- CONTENT-TYPE attribute available when the METHOD is XML or RAW.
  - If not specified and METHOD is XML, POST will have content type "text/xml."
  - If not specified and METHOD is RAW, will have content type "text/plain."
- The <MvCALL>...</MvCALL> loop terminates when the entire document has been received, or when an <MvCALLSTOP> is encountered.

See the "[MvCALL Variables](#)".

### MvCALL Variables

#### Call Object Type, Tag

If object returned is a start tag, it is separated into the following variables:

- `callvalue`  
The whole tag, including angle brackets, tag name, any attributes and attribute values.
- `callobjectelement`  
The tag name.
- `callobjectnumattributes`  
The number of attributes specified for the tag.
- `callobjectattributeN` or `callobjectattribute[ ]`  
If attributes have been specified, variables *callobjectattribute1*, *callobjectattribute2*, ... will be created, containing the *names* of the attributes as they appear in sequence. Accessible as arrays using the syntax `callobjectattribute[1]`, which has the same value as `callobjectattribute1`.
- `callobjectvalueN` and `callobjectvalue[ ]`  
If attributes have been specified for the tag, variables *callobjectvalue1*, *callobjectvalue2*, ... will be created, containing the *values* of the attributes as they appear in sequence. Accessible as arrays using the syntax `callobjectvalue[1]`, which has the same value as `callobjectvalue1`.

#### Call Object Type, Text

---

**Note:** Sequences of white space between tags (such as new lines, spaces, and tabs) are returned as text objects by MvCALL.

---

- `callobjecttype`  
Value: 'text'  
`callvalue`
- actual string of text and white space

#### HTTP Header Information

- `callnumberofheaders`  
This variable contains the number of headers that have been retrieved. The number and content of headers depends on the server software that is serving the document. The value of *callnumberofheaders* will not change during the iterations of the `<MvCALL>` loop through a particular document.
- `callreturnheaderN` and `callreturnheader[ ]`  
The headers received are placed into a separate variables *callreturnheader1*, *callreturnheader2*, and so forth. The values of these variables do not change during the iterations of the `<MvCALL>` loop through a particular document. Accessible as arrays using the syntax `callreturnheader[1]`, which has the same value as `callreturnheader1`.

---

### <MvCALLSTOP>

**Attributes**

(no attributes)

**Comments**

- Empty tag.
- Terminates MvCALL.

---

### <MvCOMMERCE>

**Attributes**

ACTION ="url\_to\_commerce\_server"

METAMETHOD="name"

FIELDS="value1,value2,...">

**Comments**

- Closed tag.
- When encountered, the Virtual Machine consults registered commerce libraries list. If found, the Virtual Machine loads the corresponding commerce library.

---

### <MvCOMMERCESTOP>

**Attributes**

(no attributes)

**Comments**

- Empty tag.
- Terminates <MvCOMMERCE>.