



Miva Empresa with Virtual Machine 4.13
Release Note
Revision 1.0

Bugs Fixed:

1. MvCALL now properly handles CDATA sections and does not terminate locally defined document types due to !ELEMENT tags within the !DOCTYPE definition.
2. The 3x configuration library will convert relative directories to absolute directories, making the following configuration fragment work as it did in the 3.x interpreter:

```
mivaroot=&[document_root]
stdmodedatadir=&[document_root]/../mivadata
```

3. MvCALL retrieving comment-style tags (e.g., "<!--", "<![CDATA[" , etc.) now correctly reports zero attributes rather than the attributes from a tag immediately preceding it.
4. mvProgram_Register_Persistent no longer causes access violations when called multiple times with the same key value.
5. XBase database interface has had some non-essential locks removed, improving NFS-mounted database access.
6. The file mivavm-vn.nn-win32.zip will contain some of the files delivered in the mivavm-vn.nn-win32.exe, but not use the Microsoft Installer.
7. Networking is correctly initialized when running ISAPI on Windows 2003.
8. The certificate authority files have been purged of expired certificates.
9. The external database routine preferred_type is called if it exists, and the variable in question has no preferred type set at update time.
10. AuthorizeNet commerce library now parses responses based on x_delim_char rather than a hard-coded ";".
11. AuthorizeNet commerce library encodes instances of the delimiter in "&#xx;" format before sending the request to AuthorizeNet.
12. Win32 IIS engine now correctly detects initialization having been done, preventing intermittent thread crashes while running under Windows 2003.
13. The built-in functions fsrename, sfrename, srename, and frename now require non-null source and target filenames, to avoid inadvertently moving the whole script directory into the data directory, or visa versa.
14. The 3x configuration library will convert relative directories to absolute directories at startup, making the following configuration fragment work as it did in the 3.x interpreter:

mivaroot=&[document_root]
stdmodedatadir=&[document_root]/../mivadata

15. The LinkPoint commerce library will truncate the number of options on an item at ten, to prevent a "too many options" error from LinkPoint.
16. MvSMTP no longer puts two extra, random characters at the end of the "X-Mailer: " header.
17. MvSMTP no longer causes a stack underflow at runtime if a SUBJECT attribute is not used.
18. The LinkPoint commerce library now includes "DECLINE:" in the error it returns when it detects an Address Verification failure.
19. The LinkPoint commerce library no longer returns intermittent, incorrect "Read Timeout" errors.
20. The LinkPoint commerce library now returns the correct r_code on successful transactions when in "SALE" mode.
21. The CyberSource ICS2 commerce library is again available on FreeBSD version 4.5.
22. The LinkPoint commerce library will return an error on failure to load the OpenSSL sharable libraries, rather than quietly failing.
23. The ORACLE database interface now correctly handles tables with more than 62 fields.
24. Time variables on Win32 platforms now consistently take the system daylight savings time variable into consideration.

New Features:

1. MvCALL now supports the HEADERS attribute. The value in this attribute will be placed after the MVM-generated HTTP headers, but before the CRLF separating the HTTP headers and the body of the http request.
2. The LinkPoint commerce library now recognizes the req_read_timeout variable, and will use the value to override the default 90 second network read timeout value.
3. The following is a new database interface routine:

preferred_type

Return the datatype preferred by the database for a given database variable.

MVD_TYPE_STRING is the default if this interface is not implemented, or for older versions of the database interface that do not support this routine.

Syntax:

int preferred_type(mvDatabaseVariable var)

Parameters:

var: mvDatabaseVariable in question.

Return Value:

Must return one of the following (defined in mivapi.h):

MVD_TYPE_NONE

MVD_TYPE_STRING

MVD_TYPE_INTEGER

MVD_TYPE_DOUBLE

4. The following are new built in functions:

tar_directory

Return information about a tar file.

Syntax:

tar_directory(filepath, location , desc)

Parameters:

filepath: File name and path within the location

location: Either "script" or "data". Anything else defaults to "data".

desc (out): An array of structures, one array element for each item in the tar file, with the following members assigned:

NAME	Text	File name.
MODE	Integer	file protection mode of file.
UID	Integer	User ID of file.
GID	Integer	Group ID of file.
SIZE	Integer	Size of file.
TIMESTAMP	Integer	Timestamp of file in seconds since 1-JAN-1970 00:00:00
TYPE	Integer	Type of file 0 = normal 1 = link 2 = symbolic link 3 = directory 4 = unsupported
LINK	Text	Link target (if link)
USER	Text	Name of user
GROUP	Text	Name of group
MAJORDEV	Integer	Major version number (if a device file)
MINORDEV	Integer	Minor version number (if a device

file)

Return Value:

Number of files in output array.

0 on error

tar_extract

Given the location of a tar file, unpacks the files into the specified directory.

Syntax:

```
tar_extract( tarfile, tarfile_loc , dir , dir_loc )
```

Parameters:

tarfile: Location of the tar file.

tarfile_loc: Either "script" or "data". Anything else defaults to "data".

dir: Directory to extract the tar file to.

dir_loc: Either "script" or "data". Anything else defaults to "data".

Return Value:

1 on success

0 on error

wget:

Syntax:

```
wget( url , filepath , location )
```

Parameters:

url: URL of file to retrieve.

filepath: Path to put retrieved file.

location: Either "script" or "data". Anything else defaults to "data".

Return Value:

HTTP retrieval code

-1 on error to get the HTTP header code.

See www.w3.org RFC 1945 or RFC 2616 for more information on the values that may be returned.

keyword_extract:

Given a string, will extract keywords from it. Skipping SGML tags and

stemming words (e.g., "running" becomes "run", but "bring" is not changed), the unique list of keywords are placed in the keywords as an array. Also, some common english words (the articles, "for" "are", etc.) are removed.

Note that there are some words may stem unexpectedly (e.g., "has" transforms to "ha").

Syntax:

```
keyword_extract( string , keywords )
```

Parameters:

string: Source string
keywords: array of keywords

Return Value:

Count of resulting keywords in the "keywords" array.

keyword_in

Performs a keyword_extract (see above) on the "string" parameter, and determines if the value in the "keywords" parameter is contained in the keyword list, and returns a boolean "1" or "0" to signify that the keyword is or is not in the string.

If the "keywords" parameter is an array of strings, checks each value in the array, and returns an array of booleans specifying whether the array element is or is not in the string.

Syntax:

```
keyword_in( keywords , string )
```

Parameters:

string: Source string
keywords: String or array of keywords to check.

Return Value:

If keywords is an array, an array of booleans specifying if a given keyword element is in the keyword list.

If keywords is a string, a boolean specifying if the given keyword is in the keyword list.

keyword_extract_merge_init

Initializes persistent storage for a multiple calls to keyword_merge_extract. Note that this persistent storage

is usable for multiple calls within a running Miva Script program, and is deleted by the VM at the end of program execution.

Syntax:

```
keyword_extract_merge_init( )
```

Parameters:

None.

Return Value:

None.

`keyword_extract_merge`

Extracts keywords as described in `keyword_extract`, but inserts them into a persistent array initialized by `keyword_extract_merge_init`. The weight value passed in is associated with the results from the current string being analyzed.

Syntax:

```
keyword_extract_merge( string, weight )
```

Parameters:

string: String to analyze for keywords.
weight: Weight to associate with the keywords found in string in the persistent array.

Return Value:

None

`keyword_extract_merge_results`

Returns results from one or more `keyword_extract_merge` calls, storing them as an aggregate in "keywords".

Syntax:

```
keyword_extract_merge_results( keywords )
```

Parameters:

keywords:

An aggregate with the following format, in keyword order:

```
keywords[ n ]:stemmed
```

The stemmed version of the keyword. All subkeys may be different text, but stemmed to this value.

keywords[n]:weight

Cumulative weight of this keyword. If a weight of 1 is used for all calls to keyword_extract_merge, this will be the total count of references to this keyword.

keywords[n]:subkey_count

Count of other references to this key.

keywords[n]:subkey[n]:keyword

Keyword (unstemmed, the stemmed value is in keywords[n]:stemmed).

keywords[n]:subkey[n]:weight

Weight of this keyword reference.

Return Value:

Number of keywords.

x509_load:

Load an X509 Certificate from the file specified by "cert"

Syntax:

x509_load(cert , x509)

Parameters:

cert: File containing an x509 certificate

x509: Index into an internal array of certificates.

Return Value:

1 on success

0 on error

x509_create:

Create an X509 Certificate from the PEM format data in "cert".

Syntax:

x509_create(cert , x509)

Parameters:

cert: PEM format certificate.

x509: Index into an internal array of certificates.

Return Value:

1 on success
0 on error

x509_verify:

Verifies that the X509 certificate specified by "x509" was issued by one of the X509 certificates (in PEM format) in "trusted_certs".

Syntax:

```
x509_verify( x509 , trusted_certs )
```

Parameters:

x509: Index into internal array of certificates.
trusted_certs: Certificates (in PEM text format) to find the x509 certificate in.

Return Value:

1 on successful find.
0 if certificate isn't in trusted_certs, or other error.

x509_rsa_publickey:

Extracts the RSA public key from the X509 specified by "x509" and stores it in "rsa".

Syntax:

```
x509_rsa_publickey( x509 , rsa )
```

Parameters:

x509: Index into internal array of x509 certificates.
rsa: Index into internal array of RSA public keys.

Return Value:

1 on success
0 on error

x509_free:

Deletes an x509 from the internal array of x509 certificates.

Syntax:

```
x509_free( x509 )
```

Parameters:

x509: Index into internal array of x509 certificates.

Return Value:

1 on success
0 on failure

crypto_md5_file:

Calculates the md5 hash of the file specified by "file", returning a non-zero value and the hash in "hash" if successful.

Syntax:

```
crypto_md5_file( file , location , hash )
```

Parameters:

file: Name of file to calculate the hash of.
location: Location (either "script" or "data", defaulting to "data" if any other value)
hash: MD5 hash value.

Return Value:

1 on success
0 on failure

rsa_sign:

Sign the data in "buffer" with the RSA private key specified by "rsa", returning the result in "signature".

This function requires OpenSSL 0.9.7 or greater.

Syntax:

```
rsa_sign( rsa, buffer , signature )
```

Parameters:

rsa: Value returned from one of the rsa_load_key routines.
buffer: Buffer to sign.
signature: Resulting signature.

Return Value:

1 on success
0 on failure

rsa_verify:

Verify the data in "buffer" with the RSA public key specified by "rsa" and the signature in "signature".

This function requires OpenSSL 0.9.7 or greater.

Syntax:

```
rsa_verify( rsa , buffer , signature )
```

Parameters:

rsa: Value returned from one of the `rsa_load_key` routines.
buffer: Buffer to sign.
signature: Signature.

Return Value:

1 on success
0 on verification failure or error.

`crypto_last_error`:

Syntax:

```
crypto_last_error()
```

Parameters:

none

Return Value:

Error text from the last SSL error, or other internal errors in the crypto suite of functions.

`crypto_last_ssl_error`:

Syntax:

```
crypto_last_ssl_error()
```

Parameters:

none

Return Value:

Integer value of the last SSL error code. Use `crypto_last_error()` instead of this routine to get the text of the last SSL error.