Active4D v6

Welcome to Active4D v6 for 4D v11. Please read the following document carefully, as there may be changes that will affect your applications.

Upgrading from v5

If you are upgrading from Active4D v5, no changes to your code should be necessary. v6 fixes bugs and adds new features.

Upgrading from v3/v4.x

I have tried as much as possible to make v6 backwards compatible with v4.5. There are very few changes to Active4D that will require changes to your code. A complete list of what has changed from v3/v4.x is contained in the rest of this document.

Changes in Active4D that MAY require a code change are in the **May Require Code Changes** section. Changes that WILL require a code change are in the **Code Change Required** section. In addition many commands have been enhanced/extended to take advantage of Unicode or to match the behavior of the 4D v11 version of a command, and those changes are documented below.

General Information

- Internally Unicode text is stored using 16-bit (2-byte) *code units*. A single Unicode *code point* (what we would usually call a character) *may* require two code units (four bytes). However, almost all modern languages are in the Unicode *Basic Multilingual Plane* (or BMP), which can be represented by a single 16-bit code unit. When referring to strings in Active4D (and in 4D v11), all indexes, lengths, etc. are in terms of 16-bit code units. Since any language you are likely to use will be in 16-bit code units, you should not have to worry about a character index falling in the middle of a 32-bit code point.

- Path limits have been increased to the maximum allowed by the operating system. The maximum path length on Mac OS X is 1024 *bytes*, on Windows 260 *characters*. The maximum length of any single segment of a path (including a filename) on either platform is 255. Note that on Mac OS X, the limit is in *bytes*, not *characters*. Filenames are internally expressed in UTF-8, which means that most non-ASCII Latin characters will take up two bytes, and most Asian characters will take up three or four bytes. These are limits imposed by the operating system, don't blame me!

- **blowfish encrypt/decrypt** use a variable length key between 1 and 56 bytes and an 8 byte IV. Both the key and IV are truncated to the maximum length or padded with spaces to the minimum length. Text to be encrypted is padded with PKCS#5 padding to an even multiple of 8 bytes before encryption. If you are encrypting and decrypting within Active4D, you don't need to know this, but if you are sharing encrypted data with other software, you need to know the encryption format.

Code Changes Required

- The URL returned by **On Request** must **NOT** be url encoded or converted to ISO-8859-1.

- The new syntax for **blowfish encrypt** is:

blowfish encrypt(text; passphrase {; iv {; charset }}) -> blob
blowfish encrypt(blob; passphrase {; iv {; charset {; * }}}) -> blob | <none>

By default, Unicode text passed to **blowfish encrypt** (key, IV and plaintext) is converted to Mac Roman for compatibility with v4.5. You can specify another character set (such as UTF-8) by passing the charset parameter, which should be a valid IANA charset name. If the name is passed and is empty, it defaults to Mac Roman. If the name is not valid, an exception is thrown.

blowfish encrypt always encrypts to a blob, because encryption actually creates a bunch of numbers which cannot reliably be converted to valid Unicode text.

blowfish encrypt can take a blob as the first parameter. If * is passed after the charset, the blob is encrypted in place and nothing is returned.

- The new syntax for **A4D Blowfish encrypt** is:

A4D Blowfish encrypt(text; passphrase {; iv {; charset }}) -> blob

This functions exactly like **blowfish encrypt**. If *charset* is not passed, is empty, or is invalid, it defaults to “mac roman” for compatibility with v4.5.

NOTE: Please note that the command now returns a BLOB, not text.

- The new syntax for **blowfish decrypt** is:

blowfish decrypt(blob; passphrase {; iv {; charset {; * }}}) -> text | <none>

blowfish decrypt takes a blob instead of text as the first parameter. If * is passed after the IV, the blob is decrypted in place and nothing is returned. Otherwise the decrypted blob is converted to Unicode text from the charset passed in, or from Mac Roman if no charset is passed.

Note that the typical use of **blowfish decrypt** as shown below will continue to work with no changes, as long as the text being decrypted can be expressed in the Mac Roman character set:

```
$b64 := base64 encode(blowfish encrypt($text; $pass; $iv))
$text := blowfish decrypt(base64 decode($b64); $pass; $iv)
```

- The new syntax for **A4D Blowfish decrypt** is:

A4D Blowfish decrypt(blob; passphrase {; iv {; charset}}) -> text

The syntax is the same as **blowfish decrypt**, but the input blob cannot be decrypted in place, it always returns text. If *charset* is not passed, is empty, or is invalid, it defaults to “mac roman” for compatibility with v4.5.

NOTE: Please note that it now takes a BLOB as the first parameter, not text.

- The new syntax for **base64 encode** is:

base64 encode(text {; charset {; * | urlSafe}) -> text
base64 encode(blob {; * | urlSafe}) -> text

By default, Unicode text passed to **base64 encode** is converted to Mac Roman for compatibility with v4.5. You can specify another character set (such as UTF-8) by passing the charset parameter, which should be a valid IANA charset name. If the name is empty, it defaults to Mac Roman. If the name is not valid, an exception is thrown.

If * is passed or *urlSafe* is passed and is True, a URL safe form of base 64 encoding is created where '+' and '/' are replaced with '-' and '_' and the "=" padding at the end is stripped. If you create URL safe encoding, be sure to do URL safe decoding as well.

base64 encode can take a blob as the first parameter, in which case the bytes of the blob are encoded directly and no charset parameter should be passed.

- The new syntax for **A4D Base64 encode** plugin command is:

A4D Base64 encode(text {; charset}) -> text

The *charset* parameter functions like the one for the **base64 encode** command,

- The new syntax for **base64 decode** is:

base64 decode(text {; charset {; * | urlSafe}}) -> text | blob
base64 decode(blob) -> blob

If passed text and no charset, **base64 decode** returns a blob.

If passed text and a charset, **base64 decode** converts from that charset to Unicode after decoding and returns text.

If * is passed or *urlSafe* is passed and is True, *text* is assumed to be in a URL safe form of base 64 encoding where '+' and '/' are replaced with '-' and '_' and the "=" padding at the end is stripped. If you do URL safe decoding, be sure the text was encoded in a URL safe form.

If passed a blob, **base64 decode** now returns a blob, because there is no guarantee that the decoded text contains valid Unicode.

- The new syntax for **A4D Base64 decode** plugin command is:

A4D Base64 decode(text {; charset}) -> text

The *charset* parameter functions like the one for the **base64 decode** command, but if it is not passed (or if passed an invalid charset name) it defaults to "mac roman" for compatibility with v4.5.

May Require Code Changes

- Active4D is always in Unicode mode. This means that all 4D commands implemented by Active4D

act like the v11 version in Unicode mode.

- If your source files contain non-ASCII characters, it is essential that they ALL are encoded in the same character set, and that character set is configured as the 'platform charset' in Active4D.ini. Otherwise those non-ASCII characters will be incorrectly converted to Unicode when the source file is read by Active4D.

- If you are using 4D as the web server host for Active4D, do NOT remove or rename the shell method A4D_GetHttpBodyCallback, unless you set a new method with **A4D Set HTTP body callback**. This method is used by the plugin to retrieve the body of a request, so without this method posted forms will not work.

- Mac OS X path lists that reference volumes other than the root volume must begin with /Volumes.

- The standard search path for config files is now:

<DB>/Active4D (standalone/server)
~/Library/Application Support/4D/com.aparajita/Active4D/conf (Mac OS X)
<user app data>/4D/com.aparajita/Active4D/conf (Windows)
<shared 4d directory>/com.aparajita/Active4D/conf
<plugin bundle>/Contents/Resources/conf

- The standard search path for libraries is now:

<DB>/Active4D (standalone/server)
<path list> (configured in Active4D.ini)
~/Library/Application Support/4D/com.aparajita/Active4D/lib (Mac OS X)
<user app data>/4D/com.aparajita/Active4D/lib (Windows)
<shared 4d directory>/com.aparajita/Active4D/lib
<plugin bundle>/Contents/Resources/lib

Path reference:

<DB>: **Get 4D folder(Database folder)**

<user app data>:
<Current user home>/Library/Application Support (Mac OS X)
{Disk}:\Documents and Settings\<Current user>\Application Data (XP)
{Disk}:\Users\Current user\AppData\Roaming (Vista/Windows 7)

<shared 4d directory>: **Get 4D folder(Licenses folder)/../**

- When running under Client, the "default" directory is no longer the Client application directory, but the directory <shared 4d directory>/com.aparajita/Active4D

- Within *Active4D.ini*, the path token <boot volume> actually refers to the volume on which the current system is found.

- The first time a native path on the root volume is created on Mac OS X (e.g. with **url to native path**), the root volume name is looked up and then cached. If you wish to rename your root volume, you must restart 4D to ensure path conversions are correct.

- The key file search path is now:

<4d shared Licenses folder>/com.aparajita/Active4D
<plugin bundle>/Contents/Resources

~/Library/Preferences or the Windows directory are no longer searched, it slows down startup too much.

- Language identifiers now follow the rules for Java identifiers, with the addition that they may contain spaces and table names may begin with a digit. See:

http://java.sun.com/j2se/1.3/docs/api/java/lang/Character.html#isJavaIdentifierStart(char)

- Date literal parsing is more rigid. Dates must contain three parts of all digits, separated by slash, space, dot or dash.

- **A4D Execute 4D request** no longer takes the web form variable names and values arrays, since posted form data is directly retrieved from the request body via a callback method.

- Regular expressions now follow the ICU syntax (http://userguide.icu-project.org/strings/regexp), which means they fully support Unicode. A regular expression pattern delimiter may be any non-alphanumeric Unicode character in the Basic Multilingual Plane. The following pattern options are available and work as before: e, i, m, s, x. The following options are no longer available: A, D, U.

Note that previously you could specify an empty pattern to match the empty space between each character. ICU does not allow this. Also note that ICU does not support named capture groups.

- Previously, if a posted form field had ":blob" suffixed to the field name, the field would be stored as a blob. This was to get around the 32K limit on text. Since that is no longer an issue, this behavior has been deprecated. All form fields are stored as text.

- When using **write blob** with text, the last argument can be a character set name (see **CONVERT FROM TEXT**) instead of *, in which case it indicates the character set of the text. If none is given, it is assumed to be UTF-8.

- Text is html encoded according to the following rules:
  - If a character is ASCII, has a named entity, and the current encoding mode asks for that entity to be encoded, it is encoded.
  - If a character is non-ASCII, the output charset is ISO-8859-1 or ISO-8859-15, the mode is kEncoding_Extended, and the character has a named entity, it is encoded.
  - Otherwise the character is passed through as is.
  - This affects everything to do with output encoding.

- When posting forms, there is no way for the browser to communicate the encoding of the page on which the form resided. It will post the form data in whatever character set the form's page used. So Active4D uses the "output charset" config option to determine how to decode form data.

- The old form of the **On Request** handler (which receives *$inQuery* and returns the query with the URL) is no longer supported.

- Response cookies are always converted to UTF-8 and url encoded. If you have already encoded the cookies, this is okay because they are converted from UTF-8 and url decoded when received.

- Query strings and request cookie values are always url decoded and converted from UTF-8.

- **url encode/decode [path/query]** converts to/from UTF-8 before encoding/decoding.

- **Mac to ISO**, **ISO to Mac**, **Win to ISO**, and **ISO to Win** are no-ops, because all text is converted to and from Unicode.

- **mac to html** is retained for compatibility, but has been renamed **html encode** to make it clearer what is happening and to remove the reference to the Mac Roman character set. Also, **html encode** will only convert non-ASCII characters if the output charset is ISO-8859-1 or ISO-8859-15.

- The **http error page** config option should be url encoded UTF-8 instead of ISO-8859-1.

- The 'content charset' config option is deprecated (and ignored) in favor of the 'output charset' config option, since the content charset should never be different than the output charset.

- **set content charset/get content charset** are now aliases for the commands **set output charset/get output charset**, since the output and content charsets have been merged.

- Because the configured output charset is used for decoding posted form data, make sure the output charset matches the charset you actually use in your pages.

- **set content charset/set output charset** can take an old-style charset constant such as *A4D Charset Mac*, or they can take an IANA charset name. Note that *A4D Charset None* is now effectively the same as *A4D Charset Mac*, because Unicode always has to be converted to a different encoding before being sent to the browser.

- Because the response buffer is Unicode, the output charset has no effect on individual **write** commands. The output charset only comes into play when the response buffer is converted to a blob before returning to 4D.

- **get platform charset** returns the IANA charset name of the current charset, instead of a number.

- The syntax of **Position** is now:

Position(inFind; inSource {; inStart {; { *; } { outLengthFound {; * }}}})

The first four parameters (through the first *) are the same as Active4D 4.5. The last two parameters are the same as the last two parameters of the v11 **Position** command, with the exception that all searches take Unicode "ignorable" characters into consideration (such as CR).

Note that when searching in reverse, the start position marks the *end* of the portion of inSource that is searched. In other words, it is as if you are looking for the last occurrence of *inFind* within **substring**(*inSource*; 1; *inStart*).

- The **Char/Get character code** command always takes UTF-16 character codes. As long as your existing usage of **Char** uses values < 127, no change is necessary.

- **open document/append document/create document** ignore the *fileType* parameter. The filename extension must be part of the *document* argument.

- The "i" ISO mode switch in **build query string** is ignored, since all text passed to the command is Unicode.

- **build query string** converts to UTF-8 instead of ISO-8859-1.

- If **get response buffer** is used with a blob, the blob receives UTF-8 encoded text, as if you had executed **TEXT TO BLOB**(buffer; blob; *UTF8 Text without length*).

- If **set response buffer** is used with a blob, the blob is assumed to contain UTF-8 encoded text with no length.

- If the *inSubject* parameter to **regex match** or **regex replace** is a BLOB, it is assumed to be in the format *UTF8 Text without length*.

- The **Char/Get character code** command will only accept Unicode code point values in the Basic Multilingual Plane, not including Unicode non-characters.

- To be accurate, **compare strings** does a bitwise comparison of the 16-bit Unicode code units in the strings, not a lexical comparison.

- **first of**, **first not of**, **last of** and **last not of** take an optional * as a last parameter. If not passed, the comparison is case and diacritical-insensitive (the backwards-compatible default). However, composed characters like "ß" will *not* be matched with non-composed equivalents like "ss". If passed, the comparison is a strict bitwise comparison, which means case and diacriticals are significant.

- An attempt to access a file outside of the web directory or a directory listed in 'safe doc dirs' will return an error code of -45 (file is locked).

- If an I/O operation returns an error number -70001 or lower, it is a standard C library error code and not a Macintosh error code. The actual C error code is *abs(error)-70000*. So for example -70001 is 1, -70002 is 2, etc. C library error codes can be found here:

http://www-numi.fnal.gov/offline_software/srt_public_context/WebDocs/Errors/unix_system_errors.html

- The **Open document** command now conforms more closely to 4D's behavior. In particular, an exclusive write lock is acquired on files opened in read/write or write mode. If no open mode is passed and a file is already open for writing, the file will be opened in read-only mode.

- The values returned by **week of year** may differ from v4.5. For a discussion of the values returned, see the "WEEK_OF_YEAR field" note at:

http://userguide.icu-project.org/datetime/calendar#TOC-Disambiguation

- **get version** no longer displays the network layer, 4D host or build flags. That info can be found in the log file.

- When using the **write gif**, **write jpg**, and **write png** commands, I'm using 4D's built in picture conversion. For jpegs the result is lower quality than what Active4D v4.5 produced. I recommend using png as the picture format for images that are not photographs.

- When scanning for circuit libraries, the currently configured library extension is matched exactly, which means that case *is* significant.

- The blob parameter to **A4D Execute BLOB** is assumed to be in the format *UTF8 Text without length* if 4D is in Unicode mode, or *Mac text without length* if 4D is in compatibility mode.

- In Unicode mode there are no restrictions on process/interprocess variable usage. In non-Unicode mode:
  - Text variables/arrays are read write.
  - String variables/arrays appear to Active4D as text variables/arrays.
  - If you declare a process/interprocess string variable/array in 4D, it is read only.
  - If you declare a process/interprocess string variable/array in Active4D, it is read write.

- **A4D Strip 4D tags** is now deprecated (it was only meant for 4D 2003) and does nothing.

- The "http error page" option in Active4D.ini must now be URL encoded UTF-8 instead of URL encoded ISO-8859-1. If the URL has no non-ASCII characters, no change is necessary.

- The default image format for images loaded from the picture library and from methods by image.a4d is PNG instead of GIF.

- **get license info** and **A4D GET LICENSE INFO** always return a Posix key file path.

Enhancements

- "session var name" no longer needs $ before name, if it is there it is ignored.

- Local variable names, method names, and library names now have a maximum length of 255 characters.

- Time literals may use spaces as separators in addition to colons, and each part of the time (hours/ minutes/seconds) may contain a single digit.

- A pointer will now be converted to a string according to the rules described for the method **a4d.utils.getPointerReferent**. This means you can pass a pointer to the **String** command, use it in a format string, append it to a string, etc.

- The Mac file type and creator type in ExtensionMap.ini is now ignored. In addition, entries may now consist only of the extension and mime type.

- Realm names and match strings are now limited to 2 billion characters.

- If the *inExpires* parameter to **set response cookie** or **set response cookie expires** is a positive number, it is considered the maximum age of the cookie in seconds. [NOT YET IMPLEMENTED]

- If you pass a second * parameter to **get response cookie expires**, you get back a number which is the maximum age of the cookie in seconds. [NOT YET IMPLEMENTED]

- The housekeeper main loop is now within 4D itself, which makes it much easier to stop and restart the housekeeper if necessary.

- **ALL RECORDS** now takes the optional autoload parameters, like **FIRST RECORD**, etc.

- A new string formatting operator has been added: **%%**. This works analogously to the % formatting operator, but it uses ICU message formatting, which in general is more powerful than either the % formatting or **param text**. Longs, doubles, strings, times, and dates are supported as format values. Booleans are also supported, but they are converted to the number 1 or 0, which is useful with choice formats. For more information, see http://icu-project.org/apiref/icu4c/classMessageFormat.html.

NOTE: When using this format, parameter numbers are zero-based.

You may also use this operator through the new **format string** command, which takes a format pattern as the first parameter, followed by one parameter for each positional placeholder in the pattern.

In general %% should be preferred over % because it works directly with Unicode, whereas the % operator has to convert all arguments to UTF-8, perform the formatting, then convert the result back to Unicode.

- Logging has been totally overhauled. Active4D now logs extensive information about its internal activities to aid in debugging.

- Logs are now kept in <database structure directory>/Logs/Active4D. The “Logs” directory is what would be returned by **Get 4D folder(Logs Folder)**. The log files are now rotated automatically when they reach 1MB in size. Seven log files are kept, with Active4D.0.log being the current log file, Active4D.1.log being the previous log file, and so on up to Active4D.6.log.

- The default log level is now 7, which logs everything.

- In addition to the log level which can be configured in Active4D.ini (or via the **set log level** command), you may also turn on debug logging, which show much more information about Active4D’s internal state. This can help if the normal logging does not uncover the source of a problem. To activate debug logging:

- Place a text file called “log_debug_level” in the log directory.
- Enter the text “debug” in the “log_debug_level” file.

- **log message** now takes an optional second boolean parameter. If true, the message is considered an error message, which means it is tagged with “[error]” in the log.

- The new syntax for **md5 sum** is:

md5 sum(text {; charset }) -> text
md5 sum(blob) -> text

By default, Unicode text passed to **md5 sum** is converted to Mac Roman for compatibility with v4.5. You can specify another character set (such as UTF-8) by passing the charset parameter, which should be a valid IANA charset name. If the name is not valid, an exception is thrown.

You may now pass a blob to **md5 sum** to calculate an MD5 hash on arbitrary data.

- **set platform charset** takes a number as it did before, or a valid IANA charset name.

- The new syntax for **A4D MD5** is:

A4D MD5(text {; charset }) -> text

By default, Unicode text passed to **A4D MD5** is converted to Mac Roman for compatibility with v4.5. You can specify another character set (such as UTF-8) by passing the charset parameter, which should be a valid IANA charset name. If the name is empty or not valid, it defaults to “mac roman”.

- **Replace string** by default uses the same algorithm as **Position**, which means composed characters like “æ” will be matched by “ae”. **Replace string** also supports the extra * parameter added by v11 to do a bitwise search. Unlike 4D, however, no searches ignore characters such as CR.

- **split string** and **slice string** use the same algorithm as **Position** when *inCaseSensitive* is false and *inPatternIsChars* is false. This means that they will correctly work with composed characters like “æ”.

- **slice string** can take a final * parameter. If passed, delimiter matching will match case and diacriticals exactly. Otherwise the default (backward-compatible) behavior is to ignore case and diacriticals.

- **find of**, **find not of**, **last of**, and **last not of** take an optional * parameter at the end. If passed, matching is case and diacritical sensitive. Otherwise the default (backward-compatible) behavior is to ignore case and diacriticals.

- If you pass * as the third parameter to **MOVE DOCUMENT**, it forces the move even if a file of the same name exists at the destination.

- **MOVE DOCUMENT** will rename folders as well as documents.

- If you pass a boolean expression as a second parameter to **redirect** and it evaluates to *True*, the response is 301 Moved Permanently instead of 303 See Other.

- If * is passed as the second parameter to **filename of**, the name returned is stripped of any extension.

- A new command, **split path**(*inPath; outDirectory; outFilename {; *}*) allows you to split a path. *outDirectory* receives the directory portion of *inPath*, *outFilename* receives the filename portion. If *inPath* ends with a directory separator, *outFilename* will be empty. If * is passed, it suppresses the trailing separator in *outDirectory*.

- A negative *where* parameter with **Substring**, **Insert string**, and **Delete string** means that many Unicode code units from the end of the string. So **Insert string**(“foo”;”bar”;-1) would insert

before the last character of “foo”, so the result would be “fobaro”. **Delete string**(“123”, -2, 1) would delete the second to last character, so the result would be “13”.

- You may pass a boolean with a format to the **String** or command or one of the **write** commands, or with a placeholder in the **param text** command. True will be treated as 1 and False will be treated as 0. So you can use formats like “Yes;;No” to transform True/False into Yes/No. Previously you would have to convert a boolean explicitly to a longint with the **Num** command to use a format is this way.

- **Count in array** can take a third parameter which is the index at which to start searching.

- A new command, **sleep**(*ticks*), is a convenience that is the equivalent of **delay process**(**current process**; *ticks*).

- Document commands can work with files >2GB in size, but because 4D text and blobs are limited to 2GB, you cannot read or write >2GB at a time.

- Added the **Get document size** command. If the document specified cannot be found, 0 is returned and OK = 0.

- Added new command, **resolve path**(*inPath {; * }*). It resolves relative paths, directory movement and aliases in *inPath* and returns an absolute full path. If * is not passed, the returned path is in URL format. If * is passed, the path is in native format. The *Error* variable is set according to the status of the path.

- All document-related commands set the *Error* variable to zero on success and an error code on failure. OK is set for commands that perform an operation and do not return an error code.

- For now the recursive delete option for **DELETE FOLDER** is disabled, it will be re-implemented in a future release.

- Previously I was not caching the location of the key file, which meant that every time you called **get license info** or **A4D Get license info**, I was looking for the key file again. This meant that the result of those commands could differ if you happened to change the key file location during execution. Now the location of the keyfile (if any) is cached at startup, so you will always get consistent results.

- All 4D 2004 and v11 named constants are available. If a named constant was renamed in v11, both the old and new names are available.

- If a 4D 2004 command was renamed in v11, both the new and old names are available.

- There is no longer a 40 character limit when querying alpha fields.

- Added the **Is field number valid** and **Is table number valid** commands.

- Added the **QUERY SELECTION WITH ARRAY** command.

- When passing an encoding as the third parameter to a **write** command, you may pass * as a synonym for **A4D Encoding All**.

- When scanning for circuit libraries and initializers, all web roots configured in Realms.ini are scanned.

- If a request is an Ajax request, **request info**{"*ajax"} = True. This allows you to quickly and easily test for Ajax requests.

- Added a new plugin command, **A4D Get IP address,** which correctly returns the first ethernet interface IP address on Mac OS X. The command **IT_MyTCPAddr** is currently broken, it returns IP addresses from non-ethernet interfaces like those created by VMWare.

- If you pass no parameters to **get utc delta**, it returns the number of minutes you subtract from UTC to get local time.

- **RowSet.newFromFile** and **RowSet.newFromData** take a new parameter at the end which is the format of the text in the document or data blob. By default the format is *Mac text without length*. If you save the file as UTF-8, you should pass *UTF8 text without length*.

- The data fed to **RowSet.newFromFile** and **RowSet.newFromData** is no longer limited to 32K.

- **a4d.json** now returns text instead of a BLOB.

- **get root** may now take a virtual host name as a parameter. If such a named virtual host exists in the virtual host table, its root directory is returned. If no such named virtual host exists, the default root directory is returned.

- The **Date** command has been enhanced. An additional syntax is now:

Date(*year; month; day*)

This allows you to construct a date programmatically without playing tricks like constructing a date string or using the idiom **Add to date**(!00/00/00!; year; month; day).

- When you call **A4D FLUSH LIBRARY**, the library is **not** flushed immediately, because libraries cannot be flushed until all Active4D interpreters stop executing. Rather, a request to flush the given library is passed to the housekeeper. The next time the housekeeper runs, it will wait for all interpreters to stop executing, then it will flush the library.

Because of this behavior, if **A4D FLUSH LIBRARY** is called again before the previous flush request is fulfilled, the second flush request will replace the first one. Ordinarily this should not be a problem, since you should not need to call **A4D FLUSH LIBRARY** at all.

If you pass "*" or "Active4D" to **A4D FLUSH LIBRARY**, the Active4D library will be flushed and immediately reloaded, but the **On Application Start** event handler will **not** be run again. You would have to execute **A4D RESTART SERVER** to accomplish that.

In general, you should not flush the Active4D library. If you are developing the event handlers in the Active4D library, you should create an auxiliary library (e.g. _active4D.a4l) and pass the Active4D event handler calls to that library. When you modify the auxiliary library, it will be reloaded and the Active4D library will remain unaffected.

If you need to flush all libraries **except** the Active4D library, you may pass "@" to **A4D FLUSH LIBRARY**.