Witango Application Server 5.0

What's New

January 8th, 2002

With Enterprise Pty Ltd
44 Miller St
North Sydney NSW 2060
Australia
PH: +61 2 9460 0500
FX: +61 2 9460 0502

www.witango.com

Table of Contents

General Introduction

Welcome! This is With Enterprise's first formal code release of the Tango technologies. Application Developers worldwide will welcome the release of Witango Application Server 5.0 and will see that the new ownership has a strong focus on:

- Optimising the performance and stability of the product suite;
- Keeping in touch with advances in technology;
- Growing the product feature set;
- Enhancing the product suite to ensure developers can continue to benefit from the advantages of using a RAD development tool.

This document examines in detail the major changes that have been made to the product suite and introduces the new meta tags and their syntax.

OVERVIEW OF MAJOR CHANGES

The major changes to features in the Witango Application Server 5.0 are as follows:

- The **threading model** has been updated. Witango Application Server 5 now runs in pre-emptive multitasking mode on operating systems that support it;
- The **mail integration** supporting international character sets in message text, MIME encoding, file attachments, custom headers, POP3, IMAP4 and SMTP;
- The **client interface** to the server has been opened up to allow for C++ and Java interfaces to facilitate integration of non-web interfaces to the server e.g. SOAP, XML-RPC;
- The **debugging facilities** have been improved to assist application developers when debugging their code. In particular, Witango Application Server 5 introduces a full meta stack Trace allowing developers to trace errors in nested meta tags;
- The **error reporting** options have been expanded to allow developers more control with error messages and associated help text;
- The **variable handling** functionality has been tightened up to:
    - o Improved performance of applications;
    - o Encourage typing of variables by developers to assist in the migration to the planned future releases of the Witango product suite.

IMPORTANT NOTES FOR BACKWARD COMPATABILITY

The release of Witango Application Server 5 includes a number of changes to default behaviours of the server. These changes have been made to tighten both the syntax and predictability of the execution of affected functions.

NOTE:
It is highly recommended that developers test their applications for these issues before upgrading their production servers.

A summary of the changes are set out below:

- Change to **default scope** in witango.ini;
- Change of request scope;
- Changes to **<@CRLF>**;
- Changes to **<@APPFILEPATH>**;

- Changes to **<@WEBROOT>**;
- Changes to **<@CHOICELIST>**;
- **Caching** is now defaulted to OFF;
- Change of calculation for **Variable Store Size;**
- Change of calculation for **Variable Expiration;**

NEW META TAGS

Witango's meta tag library has been expanded to include the following new meta tags:

- **<@DEFINE>** for creation of typed variables;
- **<@EMAIL>** for manipulation of mail integration;
- **<@EMAILSESSION>** for manipulation of mail integration;
- **<@HTTPREASONPHRASE>** for manipulation of default headers;
- **<@HTTPSTATUSCODE>** for manipulation of default headers;
- **<@ISMETASTACKTRACE>** for testing availability of meta stack;
- **<@MAKEPATH>** for the normalisation of paths;
- **<@METASTACKTRACE>** for controlling meta stack;
- **<@MIMEBOUNDARY>** to generate a MIME boundary string.

RECOMMENDATIONS FOR CHANGES TO WITANGO PROGRAMMING STYLES

Some of the new features of Witango Application Server 5 are introduced specifically to encourage application developers to use stricter programming styles that will assist them in migrating to future releases in the Witango products.

With Enterprise Pty Ltd have invested significant effort into the research and development of a new Witango Compiler. This new compiler allows the deployment of Witango application files to other environments. With Enterprise feel that Witango developers will significantly benefit from being able to deploy their code base to platforms other than the Witango Application Server as this will:

- Increase the potential client base for Witango developers to tap in to;
- Increase the market for re-usable code modules that Witango developers have already built, thus increasing the value of their intellectual property.

The first environment that the Witango Compiler will be released for is J2EE. Further environments are already in the planning phase. Developers building future applications in line with these recommendations will be rewarded when their code is compatible with the new Witango Compiler.

- It is recommended that Witango developers commence using the **<@DEFINE>** META tag to create and initialise variables, and in particular specify the variable type. Some environments that the Witango Compiler will deploy to are typed languages and the code will therefore require such definition before deployment with optimal performance.
- It is recommended that Witango developers scope all variables. This will both enhance the performance of their application and prepare their code for more seamless deployment to other more controlled environments.

Major Changes In Detail

MULTITASKING

Pre-emptive Multitasking Mode

Unlike Tango 2000 which ran as a cooperatively threaded application, the new Witango Application Server 5 now runs in a fully pre-emptive multitasking mode on operating systems that support it such as Windows 2000, Mac OS X, Linux and Solaris. This change to the server will deliver a higher level of stability and performance compared to the Tango 2000 technologies.

CONFIGURATION OF THE APPLICATION SERVER

Immediate Switching of the Witango Log File

The Witango log file will now be switched immediately after the ***SYSTEM$LOGDIR*** variable has been changed. The user will no longer need to stop the server to make the change effective.

Calculation of CACHESIZE Configuration Variable

This configuration variable is used to set how much memory is available to cache taf files and includes. To date the algorithm has not apportioned the cache between the tafs and the include files in any way. Witango Application Server 5 now utilises an algorithm to apportion the size of the Resource Cache between taf and include files in a 2:1 ratio.

Removal of ENABLETANGOUSERDOCS Configuration Variable

NON-WINDOWS PLATFORMS ONLY

The configuration variable **ENABLETANGOUSERDOCS** has been removed. All absolute, relative and virtual paths are now handled the same way on all platforms.

Locking of the witango.ini File

The configuration file ***witango.ini*** is now locked for writing operations only. Thus more than one instance of Witango Application Server 5 can use the same configuration file.

Change to VALIDHOSTS Configuration Variable

Changes to the configuration variable **VALIDHOSTS** now have immediate effect in Witango Application Server 5.

Changes to Server Related Registry Configuration Keys

WINDOWS PLATFORMS ONLY

All server-related registry configuration keys and values have been changed to reflect the changes in the server's name and ownership. The new keys are:

HKEY_LOCAL_MACHINE\SOFTWARE\WithEnterprise\WitangoServer\5.0\ *

HKEY_LOCAL_MACHINE\System\CurrentControlSet\WitangoServer

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Eventlog\Application\WitangoServer

* The rest of Witango Application Server 5 key is the same as the matching Tango 2000 registry key (after the version key - e.g. 4.0), which can be saved and restored into the new key (i.e. point at 4.0 and save, then point at 5.0 and restore using *REGEDT32.EXE*).

NOTE:

The services key shouldn't be created manually. Instead, run the following command from the command line:

witango.exe -install -c WitangoServer

The current installation code within Witango Application Server 5 uses the -c option to specify both the service display name and the internal name (all spaces in the display name will be replaced with underscores to form the internal name). This means that the system list of services will display the service name without spaces, simply as ***WitangoServer***.☐

Apache Module Name Change

The Apache module name has been changed to ***WitangoModule***. In the example, the Apache configuration file ***httpd.conf*** has to be changed to load the new plug.

Example:

WINDOWS PLATFORM

LoadModule WitangoModule c:/inetpub/scripts/wapache.dll
WitangoModule wapache.dll

NON-WINDOWS PLATFORMS

LoadModule WitangoModule /usr/libexec/httpd/wapache.so
WitangoModule wapache.so

Module Name Changes

The names of the modules listed below have been changed as follows:

MODULE FILE NAMES

Module	Old Name	Non-Windows Name	Windows Name
Application Server	t4server.exe	witangod	witango.exe
Apache plug-in	t4apache.dll	wapache.so	wapache.dll
IIS plug-in	t4iis.dll	-	wiis.dll
CGI	t4cgi.exe	wcgi	wcgi.exe

Configuration File Name Changes

The names of the configuration files listed below have been changed as follows:

CONFIGURATION FILE NAMES

Configuration File	Old Name	New Name
Application Server	t4server.ini	witango.ini
Client	t4client.ini	clients.ini

Meta Object Handlers	t4handlers.ini	handlers.ini

The default configuration entry has been changed. The new default value for the server name is ***WitangoServer***. The old one (Tango_2000_Server) can be simply renamed in the ***t4server.ini*** configuration file.

Log File Name Changes

The names of the log file listed below have been changed as shown in the following table:

LOG FILE NAMES

Log File	Old Name	New Name
System Event Log	t4events.log	witangoevents.log
Witango Application Server Log	tango.log	witango.log

OCI Shared Library Configuration Variable

A new configuration has been added to help the server locate the OCI shared library on non-Windows platforms. The variable contains the directory where the OCI shared library is located (not including the name of the shared library file). If the variable is missing from the file or is empty, the server will try to load the OCI shared library at ***$ORACLE_HOME/lib***. If none of these attempts succeeds, the OCI functionality will be disabled and a warning logged to the witangoevents.log file.

Change in Default Scope of Variables

The default scope for variables has been changed from ***User*** to ***Request***. This change was made to prevent the excessive memory consumption that occurs when variables are not explicitly scoped and when an application file doesn't return any session identifiers (e.g. **@USERREFERENCE**, etc) to the browser and a new user scope (which doesn't expire for 30 minutes) is constantly created.

VARIABLE HANDLING

Scoping of Configuration Variables

The Witango Application Server maintains an in-memory list of valid scopes for each of the Witango configuration variables. It will now generate an error if there is an attempt to set a configuration variable in an unregistered scope.

Change of Formula to Calculate the Variable Expiration

When Witango Application Server 5 is stopped, the data of the persistent scopes is now saved to a dump file with a timestamp and details of the time left for the scope before it expires. When Witango Application Server 5 is next started and reads the dump file, it calculates variable expiration time according to the following formula:

expiry time left = expiry time left at shutdown - (now - shutdown time)

This ensures that variables will expire correctly even if the server has been shutdown for a period of time.

Change to Formula to Calculate VARIABLESTORESIZE

The parameter **VARIABLESTORESIZE** of the **<@SERVERSTATUS>** meta tag now returns the size of all variables in all shared scopes and the local scope of the currently processed request. (Tango 2000 Server returned the size of all variables in all scopes).

NOTE:

Witango Application Server 5 does not add the size of the memory required by the variable store to store the variables when performing the calculation hence returning an accurate measure of the variable space.

Defining a Variable

A new **<@DEFINE>** meta tag has been introduced to allow the declaration of variables before they are used in an assignment. The new tag is described in detail later in this document. This has been implemented primarily to allow developers to begin typing and scoping their variables. This provides more predictable outcomes within an application as an error will be thrown if an attempt is made to change the type of a variable during the execution of an application without the developer explicitly purging the variable first.

Introduction of Request Scope

The name for the Local scope (i.e. Local) has been replaced with the Request.

The old name is still supported so that both names, Local and Request, may be considered as aliases. However, the name Request better reflects that the lifespan of the variables in this scope is limited by the duration of the current http request.

APPLICATION FILE

Default Data Sources No Longer Supported

The default data source defined for an entire application file is no longer supported. Each database action must now have an explicit datasource defined.

Predefined Search Argument 'Form' No Longer Supported

The undocumented predefined search argument *form* is no longer supported.

FILE PATHS

Mechanism to Resolve Paths During taf Execution

The Tango 2000 Mechanism

To resolve paths during execution the Tango 2000 Server receives two paths from the HTTP server:

- The path in the URL line (either HTTP target in case of the plug-in or PATH_INFO in case of the CGI)
- The physical path to the requested application file.

When these paths are compared, the first non-matching directory name indicates the end of HTTP server's virtual directory and the end of the virtual directory's physical path. The following example shows how the virtual directory is mapped to the physical one (the mapped parts are in the bold typeface).

Example:

Target or PATH_INFO *   /**virtual_dir**/dir4/file.taf*
Physical Path *   d:/dir1/dir2/dir3/dir4/file.taf*

Tango 2000 saves the name of the virtual directory and removes it from the paths it is trying to resolve.

Example:

when the path in the <@INCLUDE FILE="/virtual_dir/dir4/include.inc"> statement is being resolved, the part /virtual_dir/ will be removed and the rest will be added to the physical path of that virtual directory.

This make application files dependent unnecessarily on the virtual directory settings in the HTTP server.

The Witango Application Server 5.0 Mechanism

Witango Application Server 5 treats all file references the same way as the HREF parameter in HTML. If the file reference begins with a leading slash character, the requested file will be retrieved relatively to the virtual directory's root. Otherwise, the file will be read from the directory relative to the directory of the requested application file. In the case of one application file branching to another (or calling a TCF) the relative paths will be resolved with respect to the original application file not the branch or TCF call target.

Virtual directories are processed on all platforms with minimum differences. If a virtual directory is encountered in either an **<@INCLUDE>** tag or in a ***Branch*** action, it will be removed at run time before the tag or the action is processed.☐

On OS X, the **USEFULLPATHFORINCLUDE** switch controls whether the file will be treated as a full path or a relative path.

The meaning of the **<@APPFILEPATH>** is NOT changed. It still returns the directory of the currently executed taf file in URL-path format and may be changing with every new ***Branch*** action executed.

NOTE:

<@APPFILEPATH> in the TCF shows the application file path of the calling taf not the TCF file.

Examples:

The request URL:	*http://www.website.com/virtual_dir/dir1/dir2/file.taf*
The matching physical path:	*d:\phys_dir\dir1\dir2\file.taf*

The following table shows the URL paths specified in the **<@INCLUDE>** tag in a taf file and the corresponding physical paths.

<@INCLUDE> URL AND PHYSICAL PATHS

<@INCLUDE>	Physical Path
/dir1/file.inc	*d:\phys_dir\dir1\file.inc*
file.inc (relative to /dir1/dir2)	*d:\phys_dir\dir1\dir2\file.inc*
/virtual_dir/dir1/file.inc	*d:\phys_dir\dir1\file.inc*
volume/any_dir/file.inc *(Mac: USEFULLPATHFORINCLUDE == true)*	*volume:any_dir:file.inc*

NOTE:

Please see the new **<@MAKEPATH>** tag which is described in the last section of this document to assist in any conversion of Tango 2000 files.

CGI vs. Plug-In Paths

If Witango Application Server 5 is configured to use the plug-in, the extra path information cannot be passed as if it would be a CGI program.

Example:

If a CGI program is requested, the path after the request target and before the optional question mark is treated as extra path (or PATH_INFO):

http://www.website.com/cgi-bin/cgi.exe/path/file.ext?name=value

In this case, ***/cgi-bin/cgi.exe*** is the target of the request and ***/path/file.ext*** is the extra path that is passed to the CGI program (target) through the environment variable PATH_INFO. However, in case of Tango's plug-in, the URL above would be impossible to process unambiguously when Tango CGI is used:

http://www.website.com/tango-bin/t4cgi.exe/path/appfile1.taf/path/appfile2.taf?name=value

In this case there is no additional information available on where the application file ends and where the extra path begins (e.g. the first appfile1.taf may be a directory).

The current requirement is that the application files should be executed the same way regardless whether the CGI or the plug-in mechanism is used.

File Action Paths

File action path is relative to the path specified by the ***SYSTEM$ABSOLUTEFILEPREFIX*** system variable. If this variable is empty the file action path must be a fully qualified path. For example, on Windows, it must begin either with a drive specification or be in the UNC format e.g. \\computer\directory\file

Specifying relative file paths will not work as the current directory is a process-level entity and changing it from multiple worker threads will create racing conditions with unpredictable results.

DATA SOURCES

Processing of Multiple Active ODBC Statements

Under certain circumstances Tango 2000 allowed multiple active ODBC statements per connection (an active statement is defined as the one that has pending results). Witango Application Server 5 allows only one active statement per ODBC connection. Consequently, **<@DATASOURCESTATUS>** and **<@CONNECTIONS>** return similar information and the parameter **NUMCONNECTIONS** is always one for all data source types.

New Configuration Parameters

Two new configuration parameters have been added to the OS X and Linux servers to allow the system administrator to select which drivers to load. **OCILIBPATH** and **ODBCDMLIBRARY** should be configured with absolute paths to the libraries for **OCI** and **ODBC** respectively and not to an alias of the library.

Example:

OCILIBPATH=/usr/lib/OCI/8.1.7.1/lib/libclntshdylib

ODBCDMLIBRARY=/usr/lib/libiodbc.dylib

Loading Oracle OCI Dynamic Libraries

Tango 2000 Server loaded Oracle's OCI dynamic libraries on the first connection attempt. Witango Application Server 5 tries to load the libraries at the start-up time. If the library cannot be loaded it is logged to the Witango event log. If the OCI library failed to load or initialize it will not be loaded and visible in the server environment.

Errors from DBMS Servers

When SQL query execution failed under Tango 2000 Server a ***Function Sequence Error*** message would appear and not the actual error returned by the DBMS server (sometimes combined with the actual error message depending on the driver). Witango Application Server 5 server returns the correct error text reported by the DBMS driver.

APPLICATION SCOPES

No Expiry of Application Scopes

Application scopes in Tango 2000 Server expired after the **VARIABLETIMEOUT** time regardless of whether the application still existed. The Witango Application Server 5 Server does not allow application scopes to expire. The **<@RELOADCONFIG>** meta tag behaviour is not changed. The ***start-up URL*** will be called when the server is starting up and when **<@RELOADCONFIG>** is processed.

ACTIONS

Locking taf Files for Write Operations

Witango Application Server 5 allows multiple threads to read the same taf file simultaneously whilst the file is not being written to. However, it will lock an individual taf file during write operations to:

- Prevent more than one thread writing to the file at any one time
- Stop other threads reading the file while it is being updated.

When a conflict occurs (i.e. simultaneous writing and reading operations or multiple simultaneous write operations to the same taf file) then the first thread to access the file is allowed to finish it's write operation and the remaining threads are forced to wait for the operation to complete.

Deleting taf Files

Witango Application Server 5 will fail a delete file operation on a taf file if the file is open for either reading or writing by any other threads.

CHANGES TO CUSTOM HTTP HEADER GENERATION

Witango Application Server 5 always returns a complete set of HTTP response headers to the client as a special separate transaction. The owner of the client (e.g. IIS plug-in) may choose to return this error to the server or not.

NOTE:

Some browsers may show a user friendly interpretation of an HTTP error instead of the error text. If it is desirable that the error text is shown, the browser's settings must be adjusted.

The Witango Application Server 5 has two new meta tags that help to form a proper HTTP response header: **<@HTTPSTATUSCODE>** and **<@HTTPREPONSEPHRASE>**. These new meta tags are outlined in detail in last section of this document.

Change to Output of the <@CRLF> Meta Tag.

Tango 2000 used to return either CR or CRLF depending on the OS type. Witango Application Server 5 always returns CRLF as required by the HTTP RFC (RFC-2616) on all platforms. The external HTTP header file (by default ***header.htx***) should use **<@CRLF>** and should not contain any OS line breaks.

Example:

HTTP/1.1 301 Moved Permanently<@CRLF>Content-type: text/html<@CRLF><@CRLF>

META OBJECTS

Change to Meta Object Handler Loading Mechanism

The Meta Object Handler loading mechanism has been changed in Witango Application Server 5 to load handler plug-ins (DLLs) at start-up. This is no longer done on first request as it was under Tango 2000. This allows the server to verify whether a plug-in can be loaded and initialized or not on start-up. Plug-ins that fail to load or initialize are not visible in the server environment.

Several new parameters have been included in the ***handlers.ini*** file. They include:

LoadHandler	Controls whether the handler is to be loaded by the server (value 1) or not (value 0). The default value is 0 (not to load) for all handlers so as to minimise the maintenance workload of the server;
ShortName	Specifies a user-friendly name for the server that will be used for error reporting;
SupportsScanning	Indicates whether the object is capable of detecting new implementations of it's functionality or other objects of it's type;
ServerPath	Specifies the path where the server can locate handler shared library files;

VirtualMachine Specifies the path where the server can locate the Java Virual Machine library file.

Example:

In this example the COM handler will be loaded and the Bean handler will not be.

```
[Handlers]
COM=
JAVABEAN=

[COM]
LoadHandler=1
ShortName=COM / DCOM Objects
SupportsScanning=0
ServerPath=c:\\Tango2000\\W3hcm100.dll
COMAttrib=My COM parameter

[JAVABEAN]
LoadHandler=0
ShortName=JavaBeans
SupportsScanning=1
ServerPath=c:\\Tango2000\\W3hbn100.dll
BEANAttrib=My BEAN parameter
VirtualMachine=c:\\path to JVM\\jvm
```

The **<@METAOBJECTHANDLERS>** meta tag in Witango Application Server 5 will now display only those handlers whose plug-ins were successfully loaded by the server on start-up as well as static handlers (e.g. the TCF handler). Using the sample ***handlers.ini*** file shown above, the meta tag would return a two-row result array - one row for TCF and one for COM.

<@METAOBJECTHANDLERS> in Tango 2000 Server returned the description of the static handlers (i.e. TCF) and the contents of the ***t4handlers.ini*** file without checking whether the plug-ins can be loaded and initialized or not.

Concurrent Introspection Against Global Meta Objects

Tango 2000 maintained a global meta object method information cache (e.g. method names, parameters, return types, etc) that was accessible to all loaded object handlers. Moreover, the cache was flushed on every ***CloseIntrospect*** call and had to be retrieved again during an ***OpenIntrospect*** call. Augmenting this model Witango Application Server 5 now supports concurrent introspection done against the same physical object handler by securing the introspection cache for multithreaded access.

Handling of COM Objects

A meta object constructed from a VARIANT returned by a collection object (i.e. the one created when processing **<@OBJECTS>** and **<@OBJECTAT>**) doesn't have an associated object identity string. This prevents such a meta object from being introspected using the global introspection cache because there is no ID to look up the introspection information. If the returned collection object is derived from ***IDispatch*** (the kind of objects Witango supports as COM objects), then an actual CLSID of the object will be retrieved using ***IDispatch****'s* functionality. If other types of objects are returned, such as ***IUnknown***, they will receive an identity of ***IUnknown***, which is not useful in the server environment, but may be used to pass ***IUnknown*** back to the plug-in.

ERROR REPORTING

Connections from Unauthorised Clients

Attempts to connect to a server by an unauthorized client (i.e. not registered in the **VALIDHOST** parameter of the ***Witango.ini*** file) will now be reported to the witangoevents.log file.

Retrieval of Formatted Error Messages

A new error part called ***ErrorMessage*** has been added to the Witango **<@ERROR>** meta tag that allows for the retrieval of a formatted error message, in addition to the standard ***message1*** and ***message2***.

Example:

<@ERROR PART="ErrorMessage"> might return:

"The configuration variable (logdir) can not be set in the (local) scope"

Retrieval of Help on Error Messages

A new error part called ***HelpMessage*** has been added to the Witango **<@ERROR>** meta tag that allows for the retrieval of a free style optional string describing possible ways to resolve the error.

Example:

<@ERROR PART="HelpMessage"> might return:

"Consult the documentation for a list of valid scopes that can be used with this variable".

Variation to the Default Value of DEFAULTERRORFILE

The default value for the **DEFAULTERRORFILE** configuration variable has been modified to show the new parts for the **<@ERROR>** meta tag - ***ErrorMessage*** and ***HelpMessage*** - when those values are not empty.

Both ***message1*** and ***message2*** will still be available through the **<@ERROR>** tag to maintain the backward compatibility.

NOTE:

The ***HelpMessage*** text is not written into the log file.

Variation to Logging of Error Messages

There are two variations to the logging of error messages:

- If the **ERRORMESSAGE** parameter for **<@ERROR>** is not empty, it will be written into the log file instead of the usual ***message1*** and ***message2****;*
- The 255 character limit on the message length has been removed.□

Requests to Witango Application Server 5 During Start-up or Shutdown

Witango Application Server 5 no longer returns an empty screen if a request is made of the server during start-up or shutdown. It now reports one of the following three errors:

- The server is starting up;
- The server is shutting down;
- The server is not available to process regular requests.

Handling of Unrecoverable Application Errors

Previously the ***timeout.html*** file in Witango Application Server 5.0 was only returned on a recoverable exception (e.g. timed out query, memory allocation error, etc). Unrecoverable application errors resulted in a blank screen being presented back to the user. ***timeout.html*** is now also used for unrecoverable application errors. If the configuration variable **TIMEOUTHTML** is empty, the default error message ***Unrecoverable Application Error*** is returned to the user, otherwise the contents of the file will be used. The name of the configuration variable has not been changed for backward compatibility and ***timeout.html*** has been kept even though it is now used in a wider set of circumstances.

WINDOW PLATFORMS ONLY

Every time the ***Unrecoverable Application Error*** exception is caught, an entry will be added into the ***witangoevents.log*** file with the error description, the application file name and the action name, if available, that caused an unrecoverable exception. In case of a fatal exception (i.e. crash) the server tries to recover instead of restarting unless a restart is absolutely necessary.

Variation to LOGGINGLEVEL Values

The five values that can be assigned to the **LOGGINGLEVEL** configuration variable are now numeric rather than textual. The following table lists each value and describes what information is logged.

LOGGING LEVEL VALUES

Level	Information Logged
0	None.
1	Application file execution, search and post argument values.
2	Level 1 information plus application file actions.
3	Level 2 information plus generated SQL, variable and action result values.
4	Level 3 information plus ResultsHTML.

If the **LOGGINGLEVEL** value is set incorrectly in the configuration file, a warning will be reported to the witangoevents.log file and logging will be turned off. If the system variable ***SYSTEM$LOGGINGLEVEL*** is assigned an incorrect value in an application file, the request will fail with an error.

DEBUGGING

Revised Handling of DEBUGMODE Configuration Variable

The **DEBUGMODE** configuration variable no longer affects the output to the log file. It is now used exclusively to manage the debug output in the result HTML. The variable is now only evaluated once for every action executed by Witango Application Server. This means that users can no longer rely on debug in the log file.

Addition of a Meta Stack Report

When an error occurs during the processing of the result HTML, the meta stack of nesting meta tags will be reported in the log file and the log results in the debug section of the result HTML. This is known as a meta stack report.

The meta stack is reported at all log levels, except ***NOLOGGING***.

Example:

[Meta Stack] [16] *(0003):<@ASSIGN system$threadpoolsize 5>*
[Meta Stack] [16] *(0002):<@INCLUDE FILE=errors.inc>*

The numbers in parenthesis is the line number of the corresponding result HTML block. If the error occurred in one of the nested tags, as is shown in the example, the entire meta stack will be reported starting at the inner tag (i.e. the one that caused the error) and ending with the outer tag.

In this case, the line number against each line is relative to the beginning of the HTML fragment associated with the reported meta tag. The assignment meta tag that caused the error is located on the line 3 in the ***errors.inc*** file. The include meta tag is located on the line 2 of the action's result HTML.

Witango Application Server 5 can report the meta stack using two new meta tags. These meta tags are **<@ISMETASTACKTRACE>** and **<@METASTACKTRACE>**. They are defined in detail later in this document.

Additional Debug Information

Reporting debug information about external data source actions has been improved to display the action command and parameter details.

WITANGO CLIENT

Plug-In Architecture

Plug-ins and CGI have been completely redesigned and largely replaced with the Witango API code. See the *Witango.h* file for details on the API.

Cross Platform Connections

Different platforms use different symbol conventions in their path names. Windows platforms use the convention of / (forward slash) as opposed to the Unix convention of using \ (back slash). This means that you cannot use a Witango client on a Unix platform to connect to a Windows based Witango server and vice versa. However, you can connect the Witango client on a OS X box to a Linux or Solaris based server and vice versa as they are using the same path format.

ERROR_HTML

The entry ***ERROR_HTML*** in the client configuration file may be overridden by setting the ***REPORTCLIENTERROR*** entry for a particular plug-in to **YES**, **TRUE** or **1**. If it is set this way the Witango client error text will be reported regardless of whether ***ERROR_HTML*** is present or not.

Changes to Path Formats

Tango 2000's client used to convert the physical application file path reported by the web server (path_translated) to URL format (by replacing the backslashes to forward slashes on Windows. The server did the opposite operation for each request. Witango Application Server 5 sends the physical paths in the form they are provided by the web server.

Addition of CLIENTIOTIMEOUT

A new client configuration variable has been introduced to control how long the client waits for the server to respond with either the result HTML or any intermediate requests such as application or include file requests. The name of the configuration variable is **CLIENTIOTIMEOUT** and the value is expected in seconds. If the variable is missing or set to a zero, the default value will be used (which is typically quite small).

COMMAND LINE

Witango Application Server 5 can now call command line functionality on all platforms including OS X. On OS X, the script file must:

- be executable by the owner of the witangod process (i.e have rx permissions)
- use Unix syle line endings, not MacOS9-style line endings
- have the interpreter as its first line (e.g. #!/bin/sh)

For example the following script called via the external action and passed an environment variable DIR=/Users would return a list of all user's home directories on the server:

```
#!/bin/sh
ls –al $DIR
```

CRON URLS

The Tango 2000 limitation of 256 characters per cron job entry in the crontab file has been removed and URLs greater than 256 characters long can be used in the crontab file. The format of the file has not changed.

CHANGES TO EXISTING META TAGS

Removal of Undocumented Meta Tags

The undocumented and unsupported**<@PUSH>**, **<@POP>, <@SHIFT>** and **<@UNSHIFT>** meta tags are no longer recognized as valid tags.

New Parameter for <@TOKENIZE>

A new parameter **NULLTOKENS** can be used with the **<@TOKENIZE>** meta tag to recognize empty tokens. The parameter may be set to ***true*** to process empty tokens or to ***false*** to skip them. If **NULLTOKENS** is omitted, the behaviour will be compatible with that of the previous versions of server. The meta tag syntax now is:

<@TOKENIZE VALUE=text CHARS=delimiters [NULLTOKENS={true|false}] [{array parameters}]>

New Parameter for <@URL>

< @URL> has been modified to take an additional optional parameter **MAXRESULTSIZE**, which limits the size of the results retrieved from the **@URL** target server. The default value for **MAXRESULTSIZE** is 64K, the minimum value may vary, but in general is as small as 512 bytes.☐

Even though the allocated buffer will be released after the results have been processed, specifying large values may disrupt server's operations by consuming too much memory. Caution should be exercised when **MAXRESULTSIZE** is set to large (tens of megabytes) values.

@CHOICELIST OPTIONEXTRAS

The missing implementation of the **OPTIONEXTRAS** parameter in **@CHOICELIST** has now been added and operates as documented in the Meta Tags manual.

New Meta Tags

Witango’s meta tag library has been expanded to include the following new meta tags, each of which is described in detail in this section:

- **<@DEFINE>** for creation of typed variables;
- **<@EMAIL>** for manipulation of email;
- **<@EMAILSESSION>** for manipulation of email integration;
- **<@HTTPREASONPHRASE>** for manipulation of default headers;
- **<@HTTPSTATUSCODE>** for manipulation of default headers;
- **<@ISMETASTACKTRACE>** for testing availability of meta stack;
- **<@MAKEPATH>** for the normalisation of paths;
- **<@METASTACKTRACE>** for outputting meta stack;
- **<@MIMEBOUNDARY>** to generate a MIME boundary string.

<@DEFINE>

Syntax

```
<@DEFINE [NAME=]varname
[[SCOPE=]scope]
TYPE={TEXT | OBJECT | DOM | EMAIL | ARRAY}
[ROWS=rows] [COLS=columns]
>
```

Description

The meta tag creates an empty variable of the specified type in the specified scope. The type of variables created with **<@DEFINE>** cannot be changed without purging a variable first. That is, an existing TEXT variable can not be used in **<@ASSIGN>** on the left-hand side if the right-hand side variable is not TEXT.

Two optional parameters **ROWS** and **COLS** are available to create an array of a required size. They are ignored for all types, except ARRAY.

Objects created with **<@DEFINE>** are NULL objects (i.e. **@ISNULLOBJECT** returns "1") until they are reassigned to a valid object.□

Example

Given the following code defining an array and a text variable followed by assigning the test variable the value of the array:

```
<@DEFINE NAME='firstnameArray' SCOPE='user' TYPE='array' rows='1' cols='5'>
<@DEFINE NAME='firstname' SCOPE='user' TYPE='text'>
<@ASSIGN user$firstname "@@user$firstNameArray">
```

This code will throw the following error message.

File: define.taf

Position: Results

Class: Internal

Main Error Number: -627

The types of variables used in the assignment do not match.

<@EMAIL>

Syntax

```
<@EMAIL [COMMAND=]{
          STRUCTURE |□
          GETENTITYBODY |□
          GETFIELD |□
          ADDFIELD |□
          APPENDFIELD |□
          REPLACEFIELD |
          REMOVEFIELD |□
          IMPORT |□
          EXPORT}
                    NAME=emailname
                    SCOPE=scope
                    [PARTID=partid]
                    [FIELDNAME=fieldname]
                    [FIELDVALUE=fieldvalue]
                    [TYPE={XML | ARRAY*}]
                    [DECODEDATA={true | false*}]
                    [MESSAGE=messagesource]
>
```

Description

@EMAIL is one of three new meta tags (**<@MIMEBOUNDARY>**, **<@EMAIL>** and **<@EMAILSESSION>**) which have been added to allow the user to send or receive email messages using the email protocols SMTP, POP3 and IMAP4.

These tags are structured similarly to the @CIPHER with it's ACTION parameter in the sense that they perform various functions depending on the value of the first parameter of the tag and different parameters are require depending on the ACTION selected. In the case of **<@EMAIL>,** and (**<@EMAILSESSION>**, it is the COMMAND parameter. Each Tag and Command pair could be considered to be the equivalent of a simple meta tag.

PARAMETERS

Parameters	Value
COMMAND	Specifies the function to be executed. Required. (See table of for command options).
NAME	Specifies the mail to be used.
SCOPE	Specifies the scope of the email.
PARTID	Is the ID of the part of the email you are referencing
FIELDNAME	Used with command=getfield
FIELDVALUE	Used with command=getfield
TYPE	
DECODEDATA	Optional. Either true or false. If set to true the data being returned is decoded.
MESSAGE	

COMMAND OPTIONS

Command	Command Function
GETENTITYBODY	Gets the body of the entity specified.
GETFIELD	Specifies the field in the email to be returned.
ADDFIELD	Used with the structure command when creating a new mail
APPENDFIELD	Append a value to a field. Used with the structure command.
REPLACEFIELD	Replace a value of a field. Used with the structure command.
REMOVEFIELD	Remove a field. Used with the structure command.
IMPORT	Imports a text file structured as an email into a email variable.
EXPORT	Exports an email variable into a text file structured as an email.

Example

```
<@EMAIL STRUCTURE NAME=request$loopemailvar SESSIONID='POP3 Session: <@USERREFERENCE>'
TYPE='ARRAY'>

<@EMAIL GETENTITYBODY PARTID=@@request$EMPartID[<@CURROW>,1]
NAME=request$loopemailvar>

<@EMAIL GETFIELD NAME=request$loopemailvar FIELDNAME="subject">
```

<@EMAILSESSION>

Syntax

```
<@EMAILSESSION [COMMAND=] {
        OPEN |□
        CLOSE |□
        LIST |□
        RETRIEVE |□
        SEND |□
        DELETE}
                SESSIONID=sessionid
                PROTOCOL={SMTP | POP3 | IMAP4}
                SERVER=server-address
                [PORT=server-port]
                [USERNAME=username]
                [PASSWORD=password]
                [MAILBOX=mailbox]
                [MODE={COMMIT | ROLLBACK*}]
                [FIELDS=field-list]
                [MESSAGEID=mesageid]
                [NAME=emailname]
                [SCOPE=emailscope]
>
```

Description

This is one of three new meta tags (**<@MIMEBOUNDARY>**, **<@EMAIL>** and **<@EMAILSESSION>**) which have been added to allow the user to compose and manipulate an email message to send or receive email messages using such email protocols as SMTP, POP3 and IMAP4 .

These tags are structured similarly to the @CIPHER with it's ACTION parameter in the sense that they perform various functions depending on the value of the first parameter of the tag and different parameters are require depending on the ACTION selected. In the case of **<@EMAIL>,** and (**<@EMAILSESSION>**, it is the COMMAND parameter. Each Tag and Command pair could be considered to be the equivalent of a simple meta tag.

PARAMETERS

Parameters	Value
COMMAND	Specifies the function to be executed. Required. (See table of for command options).
SESSIONID	Optional. Defined using the open command and used by the other commands to identify the open session.
PROTOCOL	The protocol being used. (SMTP, POP3 or IMAP4) to make the connection with the mailserver.
SERVER	Required when using the open command. Host name (developer.witrango.com) or IP address (196.142.203.1) of the POP/SMTP/IMAP4 server.
PORT	Optional. Defaults to the standard port 110. Used with the open command.
USERNAME	Optional. If no user name is specified, the connection is made anonymously.

Parameters	Value
PASSWORD	Optional. Password corresponds to the user name.
MAILBOX	Optional. Specifies the mail box on the server to be used.
MODE	Required when using the close command. Used to either commit or rollback the changes to the mail account since the session was opened (delete read messages)
FIELDS	Optional. Specifies which fields are to be used with the chosen command.
MESSAGEID	Required with the delete and retrieve commands to identify the desired message. (COMMAND=delete MESSAGEID=12345)
NAME	Specifies Email Name.
SCOPE	Specifies Email Scope.

COMMAND OPTIONS

Command Option	Command Function
OPEN	Open an email session. To perform an interactions with the mail server this command needs to be use first. This command requires values for sessionid, protocol, server.
CLOSE	Closes the email session. This command requires values for sessionid and mode.
LIST	Returns the list of messages currently in the mail account.
RETRIEVE	Retrieve a specific message as specified by the messageid parameter.
SEND	Sends a mail that has been constructed.
DELETE	Deletes mail from mail server. This command needs the session id and the message id.

Example

<@EMAILSESSION OPEN PROTOCOL="POP3" SESSIONID="POP3 Session: <@USERREFERENCE>" SERVER="10.1.2.10"   USERNAME="username" PASSWORD="password">

<@EMAILSESSION LIST SESSIONID="POP3 Session: <@USERREFERENCE>">

<@EMAILSESSION RETRIEVE NAME=request$loopemailvar MESSAGEID='<@VAR request$messageList[@@local$loopcnt,1]>' SESSIONID="POP3 Session: <@USERREFERENCE>">

<@HTTPSTATUSCODE>

Syntax

<@HTTPSTATUSCODE>

Description

<@HTTPSTATUSCODE> is used in conjunction with **<@HTTPREASONPHRASE>** to form a proper HTTP response header.

<@HTTPSTATUSCODE> evaluates to either 200 or 500. The primary use of this meta tag is in the default header returned by Witango Application Server 5.

When a custom HTTP header is returned, it can be formed using **<@HTTPSTATUSCODE>** and **<@ HTTPREASONPHRASE >.**

Example

HTTP/1.1 <@HTTPSTATUSCODE> <@HTTPREASONPHRASE><CRLF>... the rest of the custom header ...

<@ HTTPREASONPHRASE >

Syntax

<@HTTPREASONPHRASE>

Description

<@ HTTPREASONPHRASE > is used in conjunction with **<@HTTPSTATUSCODE>** to form a proper HTTP response header.

<@ HTTPREASONPHRASE > reports the matching status phrase: *OK* or *Application Server Error*. The primary use of this meta tag is in the default header returned by Witango Application Server 5.

Example

HTTP/1.1 <@HTTPSTATUSCODE> <@HTTPREASONPHRASE><CRLF>... the rest of the custom header ...

<@ISMETASTACKTRACE>

Syntax

<@ISMETASTACKTRACE>

Description

Returns 1 if the meta stack trace is available (i.e. if the error occurred while processing the results HTML as opposed an action error) otherwise 0 is returned.

Example

<@ISMETASTACKTRACE>

<@METASTACKTRACE>

Syntax

<@METASTACKTRACE>

Description

Returns a two-dimensional array representing the meta stack trace. The first column of this array is a line number on which the error occurred and the second one is the meta tag that caused an error. **<@METASTACKTRACE>** takes the usual array formatting parameters that allow the user to change the array's appearance.□

Example

<@METASTACKTRACE>

<@MAKEPATH>

Syntax

<@MAKEPATH [PATH1=]path1 [[PATH2=]path2] [TYPE={URL|FILESYSTEM}]>

Description

In its simplest form, when only one path is provided, the path is normalized - all path delimiters are converted to the requested type (URL path or physical path) and the end of the path is appended with a path delimiter character.

When two paths are provided, each one of them will be normalized (except that the second path will not be appended with the delimiter, so that a file name can be used) and combined into a single path, returned by the meta tag. For example

Example

Non-Windows:

<@MAKEPATH "c:\inetpub/wwwroot/" "<@APPFILEPATH>" TYPE="FILESYSTEM">

will evaluate to

c:\inetpub\wwwroot\appfilepath\

Windows:

<@MAKEPATH "c:\inetpub/wwwroot/" "<@APPFILEPATH>filename.ext" TYPE="FILESYSTEM ">

will evaluate to

c:\inetpub\wwwroot\appfilepath\filename.ext.

Note: *There is no trailing slash in this case.*

<@MIMEBOUNDARY>

Syntax

<@MIMEBOUNDARY LEVELID=levelid [BOUNDARY=boundary]>

Description

The meta tag generates a MIME boundary string that can be used when composing multipart messages. If the parameter **BOUNDARY** is omitted (recommended), the value of the request scope identifier will be used to generate boundary. The **LEVELID** parameter is a number that identifies the boundary level (if a multilevel message is being composed).

Example

The resulting boundary will take the following form where the first number is the level ID and the following alphanumeric sequence is the request scope identifier at the time when the **<@MIMEBOUNDARY>** was being processed:

<@MIMEBOUNDARY LEVELID=1>

would return:

----=_MimePart_0001___33A8F4D74DE30EF93CBEFAA4