



Witango Application Server v5.5

What's New

August 2004

Witango Technologies Pty Ltd
Level 1,
44 Miller Street,
North Sydney, NSW, 2060
Australia

Telephone: +612 9460 0500
Fax: +612 9460 0502

Email: info@witango.com
Web: www.witango.com

Table of Contents

I	Table of Contents	I
2	Introduction	I
	What's New	I
	New Meta Tags	2
	Modified Meta Tags	3
	Important Notes for Migration of Applications to v5.5	3
3	New Meta Tags	5
	<@IFNOTEMPTY> <@ELSE> </@IF>	6
	<@ELSEIFNOTEMPTY>	7
	<@WHILE> </@WHILE>	8
	<@LDAPSEARCH>	9
	<@LDAPADD>	11
	<@LDAPMODIFY>	13
	<@LDAPDELETE>	15
	<@DOMSEARCH>	17
	<@XSLT>	19
	<@SLEEP>	22
4	Modified Meta Tags	23
	<@CIPHER>	24
	<@EMAIL>	27
	<@CLEARERRORS>	28
	<@TOKENIZE>	29
	<@REPLACE>	30
	<@SERVERSTATUS>	31
	<@PURGECACHE>	32
	Description	32
	Examples	32
	<@CGIPARAM>, <@HTTPATTRIBUTE>	34
	Syntax	34
	<@VAR>	37
	Shorthand for XPATH and XPOINTER	37
	<@DOM>, <@DOMINSERT>, <@DOMREPLACE>, <@DOMDELETE>, <@ELEMEN-	

TATTRIBUTE>, <@ELEMENTNAME>, <@ELEMENTVALUE>	38
Change	38
Example	38
5 New Features	41
XPath	42
Web Services	43
Enabling Web Services	43
Register a file extension	44
Creating a WSDL file for your tcf	44
Configuring theWitango Server for Web Services	49
SOAP related variables and parameters	49
Server Configuration Changes	50
Client.ini	50
Witango.ini	50
Data Source Pool	51
Document and Include cache	51
DOM variables	51
Witango Event log	51
JavaScript Engine	52
Improved error messages in web server plug ins.	52
Web Server Module Name Changes	52
Changes to Encoding	53
Changes	53
Conversion to 5.5	54
Licensing File Changes	55
Entering a new License File	55
Changing a License File	55

Introduction

Overview of What's New in Witango v5.5 Application Server

In the v5.5 release of the Witango server we have continued to focus on:

- Optimising the performance and stability of the product suite;
- Keeping in touch with advances in technology;
- Growing the product feature set; and
- 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 changes that have been made to the Witango 5.5 Server release, the introduction of the ability to support Web Services and introduces several new meta tags and their syntax.

What's New

Support for Web Services

Witango Server 5.5 incorporates SOAP document literal web services based on a WSDL document. The Witango Server 5.5 is capable of accepting a SOAP request that is mapped to a tcf/wSDL file and will respond with a SOAP response without the user having to write a line of code. This means that any TCF file can now be exposed as a web service.

A utility taf has been written that will automatically create a WSDL file from a tcf.

For more information, see “Web Services” on page 45.

Support for DOM2, XPath & XSLT

All the existing XML related meta tags now support DOM2, XPath and XSLT.

`<@DOMSEARCH>` has been added to allow searching across DOMs.

`<@XSLT>` have been added to enable XSL transformations.

Data Source Pool Changes

There have been changes made to the data source pool functionality within the Witango Server. These have been made to improve Server

optimisation and performance. It is not envisaged that these changes will result in any migration issues for users.

Document Cache Changes

There have been changes made to the document caching functionality within the Witango Server to improve both performance and stability.

This has changed the way `<@PURGECACHE>` works such that the entire cache is now only purged when explicitly stated. This will constitute a migration issue for some users which is explored later in this document.

For more information, see “Document and Include cache” on page 65.

Encoding Changes

There have been major changes to how the encoding in the Witango Server works. In particular:

- The default encoding is now `NONE` ;
- Encoding methods has now been expanded to include `META` and `HTML`.
- Functionality has changed for `METAHTML`;
- Functionality has changed for `MULTILINE`;
- Functionality has changed for `MULTILINEHTML`.

These changes ensure that the methods now describe what encoding to apply to a string.



Caution This issue **MUST** be dealt with when migrating your site to Witango Server 5.5 or unexpected results may occur.

For more information, see “Changes to Encoding” on page 67.

New Meta Tags

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

`<@IFNOTEMPTY>` `<@ELSE>` `</@IF>` on page 6.

`<@ELSEIFNOTEMPTY>` on page 7.

`<@WHILE>` `</@WHILE>` on page 8.

`<@LDAPSEARCH>` on page 9.

`<@LDAPADD>` on page 11.

<@LDAPMODIFY> on page 13.

<@LDAPDELETE> on page 15.

<@DOMSEARCH> on page 17.

<@XSLT> on page 19.

<@SLEEP> on page 22.

Modified Meta Tags

<@CIPHER> on page 24.

<@EMAIL> on page 27.

<@CLEARERRORS> on page 28.

<@TOKENIZE> on page 29.

<@REPLACE> on page 30.

<@SERVERSTATUS> on page 31.

<@PURGECACHE> on page 32.

<@CGIPARAM>, <@HTTPATTRIBUTE> on page 34.

<@VAR> on page 37.

<@DOM>, <@DOMINSERT>, <@DOMREPLACE>,
<@DOMDELETE>,
<@ELEMENTATTRIBUTE>,<@ELEMENTATTRIBUTES>,
<@ELEMENTNAME>, <@ELEMENTVALUE> on page 38.

Important Notes for Migration of Applications to v5.5

The release of Witango Application Server 5.5 includes a number of changes to the default behaviours of the server . These changes have been made to tighten both the syntax and predictability of the execution of affected functions.



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

The major changes that will affect compatibility on the Witango Server 5.5 and which should be addressed in the migration of any application are:

- 1 The **default encoding method** for the server is now `NONE`. The previous setting was `HTML` in a result actions and `NONE` in other actions. For more information, see “Changes to Encoding” on page 67.
- 2 **MULTILINEHTML** encoding now does both `MULTILINE` and `HTML` encoding. In previous versions of the server `MULTILINEHTML`

would do a `MULTILINE` encode but not perform a `HTML` encoding. For more information, see “Changes to Encoding” on page 67.

- 3 **MULTILINE** encoding now does only `MULTILINE` encoding. In previous versions of the server `MULTILINE` would do a `MULTILINE` encode and a `HTML` encoding. For more information, see “Changes to Encoding” on page 67.
- 4 **METAHTML** encoding now does both `META` and `HTML` encoding. In previous versions of the server `METAHTML` would do a meta tag encode (evaluate the meta tags) but not perform a `HTML` encoding. For more information, see “Changes to Encoding” on page 67.
- 5 **<@CGIPARAM>** has been deprecated and replaced with `<@HTTPATTRIBUTE>`. Users should replace all `<@CGIPAR AM>` occurrences with `<@HTTPATTRIBUTE>` to ensure compatibility with future versions of the server. For more information, see “<@CGIPARAM>, <@HTTPATTRIBUTE>” on page 34.
- 6 **<@PURGECACHE>** now only purges an entire `CACHE` when explicitly stated. For more information, see “<@PURGECACHE>” on page 32.
- 7 If you have overridden the setting of **ENCODERESULTS** in other scopes in your application files you will need to replace each instance of its use with `ENCODEHTTPRESPONSE`. For more information, see “Vitango.ini” on page 64.

New Meta Tags

2

New Meta Tags added with Witango 5.5 functionality

This chapter outlines syntax and describes functionality of the following new Meta Tags which have been included in Witango 5.5:

<@IFNOTEMPTY> <@ELSE> </@IF> on page 6

<@ELSEIFNOTEMPTY> on page 7

<@WHILE> </@WHILE> on page 8

<@LDAPSEARCH> on page 9

<@LDAPADD> on page 11

<@LDAPMODIFY> on page 13

<@LDAPDELETE> on page 15

<@DOMSEARCH> on page 17

<@XSLT> on page 19

<@SLEEP> on page 22

<@IFNOTEMPTY> <@ELSE> </@IF>

Syntax

```
<@IFNOTEMPTY VALUE=value>
    trueSubstitutionText
[<@ELSE>
    falseSubstitutionText]
</@IF>
```

Description

If the value specified in **VALUE** is an empty string, <@IFNOTEMPTY VALUE=value><@ELSE></@IF> includes trueSubstitutionText otherwise, it includes falseSubstitutionText.

The **VALUE** attribute value may be a meta tag or literal value (though makes little sense to use a literal value). The <@ELSE> portion is optional.

The trueSubstitutionText and falseSubstitutionText may include other <@IF>, <@IFEMPTY >, <@IFNOTEMPTY >, and <@IFEQUAL> meta tags.

Example

```
<@IFNOTEMPTY VALUE="<@HTTPATTRIBUTE NAME='USERNAME'>">
    <@IF "<@HTTPATTRIBUTE NAME='USERNAME'>=Admin">
        <H3>Administrator Options</H3>
        ...administrator options...
    <@ELSE>
        <H3>Hi, <@HTTPATTRIBUTE NAME="USERNAME">!!</H3>
        Here are your options
        ...user options...
    </@IF>
<@ELSE>
    Here are the guest options:
    ...guest options...
</@IF>
```

This example returns different HTML based on the value of <@HTTPATTRIBUTE NAME="USERNAME">.

<@ELSEIFNOTEMPTY>

Syntax

```
[<@ELSEIFNOTEMPTY VALUE=expr>  
    trueSubstitutionText]
```

If the value specified in **VALUE** is one or more bytes in length, <@ELSEIFNOTEMPTY VALUE=value> includes trueSubstitutionText. The **VALUE** attribute value may be a meta tag or literal value (though makes little sense to use a literal value).

The trueSubstitutionText may include other <@IF>, <@IFEMPTY>, <@IFNOTEMPTY>, <@ELSEIFNOTEMPTY>, and <@IFEQUAL> meta tags.

<@WHILE> </@WHILE>

Syntax

```
<@WHILE [EXPR=expr] [PUSH=push]>
</@WHILE>
```

Description

The purpose of the <@WHILE></@WHILE> pair is to provide simple *while* loop functionality.

<@WHILE> executes the HTML and meta tags between the opening and closing tags for each iteration of the loop. This means that all the HTML between the tags is sent to the Web server while the expression returns true.

Inside a while loop, <@CURREN> can be used to get the value of the index which indicates the number of iterations the while loop has executed.

EXPR is evaluated for each iteration of the loop and determines when the while loop will exit. The loop exits when the expression evaluates to false. If no **EXPR** is specified the loop exits without executing the HTML and meta tags between the opening and closing tags.

PUSH allows the sending of data to the client after the specified number of iterations have taken place.

This tag must appear in pairs and cannot span multiple actions.

Example

```
<@ASSIGN request$Counter "0">
<@WHILE EXPR="<@VAR request$Counter><5" PUSH=2>
    The counter equals @@request$Counter<BR>
    <@ASSIGN request$Counter "<@CALC
    EXPR='@@request$Counter+1'>">
</@WHILE>
```

This example outputs the following:

The counter equals 0

The counter equals 1

The counter equals 2

The counter equals 3

The counter equals 4

<@LDAPSEARCH>

Syntax

```
<@LDAPSEARCH SERVER=serveraddress
    [PORT=port]
    [PROTOCOL=ldap]
    [USERNAME=username]
    [PASSWORD=password]
    DN=distinguishedname
    FILTER=filter
    [FILTERSCOPE=ldapscope]
    ATTRIBUTES=ldapattribute
    [ATTRDELIM=attributeDelimiterString]
    [VALUEDELIM=valueDelimiterString]
    [TIMEOUT=timeout]
    [RETURNTYPE=array|dom ]
>
```

Description

LDAP is short for Lightweight Directory Access Protocol. It is an open-standard protocol for accessing and modifying information directories. A directory is a database of information that has been optimized for information retrieval. LDAP is based on the standards contained within the X.500 standard, but is significantly simpler. LDAP allows you to access directory services across a TCP/IP network using string representations of the attributes and values in the directory. Witango supports the LDAP 3 protocol.

<@LDAPSEARCH> will request a search of a LDAP directory server and return the results as an array or DOM.

Parameter	Required	Description	Default
SERVER	Yes	Defines the host name or ip address of the ldap server to connect to.	
PORT		Defines the port number that the ldap server is listening on.	389 for ldap 636 for ldaps
PROTOCOL		Indicates whether you wish to use ldap or secure ldap (ldaps).	ldap

Parameter	Required	Description	Default
USERNAME		Specifies the username to log into the directory with.	
PASSWORD		Specifies a password for a username.	
DN	YES	Specifies the Distinguished name of the search criterial.	
FILTER		Specifies a filter to run on the ldap server before retrieving the results of the search objectclass=*	
FILTERSCOPE		Specifies the how to apply the filter on the ldap server.	
ATTRIBUTES		Defines which attributes to retrieve from the ldap server.	
ATTRDELIM		Defines which character to use as a delimiter between the attributes when multiple attributes are retrieved from the ldap server.	
VALUEDELIM		Defines which character to use as a delimiter between the values of an attribute retrieved from the ldap server.	
TIMEOUT		Specifies a timeout for the query. This is how long to wait for a response from the server.	
RETURNTYPE		Defines how the Witango server will wrap the results from the ldap server. The results can be stored as an ARRAY or as a DOM variable.	

Examples

Return the LDAP search results as an array

```
<@LDAPSEARCH SERVER='192.168.0.1' PORT='389'
PROTOCOL='ldap' USERNAME='cn=Manager,dc=example,dc=com'
PASSWORD='secret' DN='dc=example,dc=com'
FILTER='objectclass=*' FILTERSCOPE='sub' ATTRIBUTES='cn'
ATTRDELIM=';' VALUEDELIM=',' STARTROW='3' MAXROWS='5'
TIMEOUT='3' RETURNTYPE="ARRAY">
```

Return the LDAP search results as a DOM

```
<@LDAPSEARCH SERVER='192.168.0.1' PORT='389'
PROTOCOL='ldap' USERNAME='cn=Manager,dc=example,dc=com'
PASSWORD='secret' DN='dc=example,dc=com'
FILTER='objectclass=*' FILTERSCOPE='sub' ATTRIBUTES='cn'
ATTRDELIM=';' VALUEDELIM=',' SORT='cn' MAXROWS='2'
TIMEOUT='3' RETURNTYPE="DOM">
```

<@LDAPADD>

Syntax

```

<@LDAPADD SERVER=serveraddress
    [PORT=port]
    [PROTOCOL=ldap]
    [USERNAME=username]
    [PASSWORD=password]
    DN=distinguishedname
    FILTER=filter
    ATTRIBUTES=ldapattribute
    [ATTRDELIM=attributeDelimiterString]
    [VALUEDELIM=valueDelimiterString]
    [TIMEOUT=timeout]
>

```

Description

LDAP is short for Lightweight Directory Access Protocol. It is an open-standard protocol for accessing and modifying information directories. A directory is a database of information that has been optimized for information retrieval. LDAP is based on the standards contained within the X.500 standard, but is significantly simpler. LDAP allows you to access directory services across a TCP/IP network using string representations of the attributes and values in the directory. Witango supports the LDAP 3 protocol.

<@LDAPADD> will add a node to a LDAP directory server based on the DN and attributes provided.

Parameter	Required	Description	Default
SERVER	Yes	Defines the host name or ip address of the ldap server to connect to.	
PORT		Defines the port number that the ldap server is listening on.	389 for ldap 636 for ldaps
PROTOCOL		Indicates whether you wish to use ldap or secure ldap (ldaps).	ldap
USERNAME		Specifies the username to log into the directory with.	

<@LDAPADD>

Parameter	Required	Description	Default
PASSWORD		Specifies a password for a username.	
DN	YES	Specifies the Distinguished name of the search criterial.	
FILTER		Specifies a filter to run on the ldap server before retrieving the results of the search objectclass=*	
ATTRIBUTES	YES	Defines which attributes to add to the ldap server.	
ATTRDELIM		Defines which character to use as a delimiter between the attributes when multiple attributes are retrieved from the ldap server.	
VALUEDELIM		Defines which character to use as a delimiter between the values of an attribute retrieved from the ldap server.	
TIMEOUT		Specifies a timeout for the query. This is how long to wait for a response from the server.	

Example

```
<@LDAPADD SERVER='192.168.0.1' PORT='389' PROTOCOL='ldap'
USERNAME='cn=Manager,dc=example,dc=com' PASSWORD='secret'
DN='cn=IT,dc=example,dc=com' ATTRIBUTES='cn=Elle
Dap;sn=Dap;objectClass=person' ATTRDELIM=';' VALUEDELIM=', '
TIMEOUT='3' >
```

<@LDAPMODIFY>

Syntax

```
<@LDAPMODIFY SERVER=serveraddress
    [PORT=port]
    [PROTOCOL=ldap]
    [USERNAME=username]
    [PASSWORD=password]
    DN=distinguishedname
    FILTER=filter
    ATTRIBUTES=ldapattribute
    [ATTRDELIM=attributeDelimiterString]
    [VALUEDELIM=valueDelimiterString]
    [TIMEOUT=timeout ]
>
```

Description

LDAP is short for Lightweight Directory Access Protocol. It is an open-standard protocol for accessing and modifying information directories. A directory is a database of information that has been optimized for information retrieval. LDAP is based on the standards contained within the X.500 standard, but is significantly simpler. LDAP allows you to access directory services across a TCP/IP network using string representations of the attributes and values in the directory. Witango supports the LDAP 3 protocol.

<@LDAPMODIFY> will modify a node to a LDAP directory server based on the DN and attributes provided.

Parameter	Required	Description	Default
SERVER	Yes	Defines the host name or ip address of the ldap server to connect to.	
PORT		Defines the port number that the ldap server is listening on.	389 for ldap 636 for ldaps
PROTOCOL		Indicates whether you wish to use ldap or secure ldap (ldaps).	ldap
USERNAME		Specifies the username to log into the directory with.	

Parameter	Required	Description	Default
PASSWORD		Specifies a password for a username.	
DN	YES	Specifies the Distinguished name of the search criterial.	
FILTER		Specifies a filter to run on the ldap server before retrieving the results of the search objectclass=*	
ATTRIBUTES	YES	Defines which attributes to update on the ldap server.	
ATTRDELIM		Defines which character to use as a delimiter between the attributes when multiple attributes are retrieved from the ldap server.	
VALUEDELIM		Defines which character to use as a delimiter between the values of an attribute retrieved from the ldap server.	
TIMEOUT		Specifies a timeout for the query. This is how long to wait for a response from the server.	

Example

```
<@LDAPMODIFY SERVER='192.168.0.1' PORT='389' PROTOCOL='ldap'
USERNAME='cn=Manager,dc=example,dc=com' PASSWORD='secret'
DN='cn=ElleDap,dc=example,dc=com' ATTRIBUTES='sn=Dap'
ATTRDELIM=';' VALUEDELIM=',' TIMEOUT='3' >
```

<@LDAPDELETE>

Syntax

```

<@LDAPDELETE SERVER=serveraddress
    [PORT=port]
    [PROTOCOL=ldap]
    [USERNAME=username]
    [PASSWORD=password]
    DN=distinguishedname
    [TIMEOUT=timeout ]
>

```

Description

LDAP is short for Lightweight Directory Access Protocol. It is an open-standard protocol for accessing and modifying information directories. A directory is a database of information that has been optimized for information retrieval. LDAP is based on the standards contained within the X.500 standard, but is significantly simpler. LDAP allows you to access directory services across a TCP/IP network using string representations of the attributes and values in the directory. Witango supports the LDAP 3 protocol.

`LDAPDELETE` will delete a node on an LDAP directory server based on the DN provided.

Parameter	Required	Description	Default
SERVER	Yes	Defines the host name or ip address of the ldap server to connect to.	
PORT		Defines the port number that the ldap server is listening on.	389 for ldap 636 for ldaps
PROTOCOL		Indicates whether you wish to use ldap or secure ldap (ldaps).	ldap
USERNAME		Specifies the username to log into the directory with.	
PASSWORD		Specifies a password for a username.	
DN	YES	Specifies the Distinguished name of the search criterial.	
TIMEOUT		Specifies a timeout for the query. This is how long to wait for a response from the server.	

<@LDAPDELETE>

Example

```
<@LDAPDELETE SERVER='192.168.0.1' PORT='389' PROTOCOL='ldap'  
  USERNAME='cn=Manager,dc=example,dc=com' PASSWORD='secret'  
  DN='cn=Elle Dap,dc=example,dc=com' TIMEOUT='3' >
```

<@DOMSEARCH>

Syntax

```
<@DOMSEARCH OBJECT=variable [SCOPE=scope]
[XPOINTER=xpointer] [XPATH=xpath]>
```

Description

This tag is used to search for XML within a document instance.

The **OBJECT** attribute (and optional **SCOPE** attribute) define the variable which contains the document instance.

The **XPOINTER/XPATH** attributes provides the search criteria in the document instance to be matched.

<@DOMSEARCH> can be used to search a DOM variable based on either an xpointer or xpath. It is not possible to use both. The result of <@DOMSEARCH> is a DOM of the nodes that match the search criteria.



Note XPath and XPointer are specified by the World-Wide Web Consortium. For detailed information on XPath, see the W3C website at www.w3.org/TR/xpath. For detailed information on XPointer, see the W3C website at www.w3.org/TR/xptr-framework.

EXAMPLE

The following example extracts the elements named last, which contain employee last names, from the employeesimple.xml file, and displays the names.

Based on the following DOM:

```
<XML>
  <DIV>
    <P>Paragraph 1</P>
    <P>Paragraph 2</P>
  </DIV>
  <DIV>
    <P>Paragraph 3</P>
    <P>Paragraph 4</P>
  </DIV>
</XML>
<@assign
  name="mydom"
```

<@DOMSEARCH>

```
value="<@DOM VALUE='
    <XML>
    <DIV>
    <P>Paragraph 1</P>
    <P>Paragraph 2</P>
    </DIV>
    <DIV>
    <P>Paragraph 3</P>
    <P>Paragraph 4</P>
    </DIV>
    </XML>
'>
```

">

<@DOMSEARCH Object='mydom' ELEMENT='child(1).child(2) '>

Returns:

```
<P>Paragraph 2</P>
```

<@DOMSEARCH Object='mydom' XPATH='DIV/P'>

Returns:

```
<root>
<P>Paragraph 1</P>
<P>Paragraph 2</P>
<P>Paragraph 3</P>
<P>Paragraph 4</P>
</root>
```

<@XSLT>

Syntax

```
<@XSLT OBJECT=dom [SCOPE=scope]
  STYLESHEET=xslstylesheet [ENCODING=encoding]>
```

Description

<@XSLT> takes a DOM variable and applies an Extensible Stylesheet Language Transformation (XSLT) to it. The XSLT transforms the XML to another format based on the rules in the Extensible Stylesheet Language (XSL) stylesheet. The results of the <@XSLT> meta tag is an XML string.

Example

To illustrate how <@XSLT> will transform a DOM into an XML string based on a style sheet, we first need to create a DOM variable and an xsl style sheet.

The following XML that has been converted into a DOM variable named MyDom:

```
<@assign request$MyDom value='<@dom value="
  <s1 title="s1 foo">
    <s2 title="Foo">
      <p>Hello</p>
    </s2>
    <s2 title="Bar">
      <p>olleH</p>
    </s2>
  </s1>
">'>
```

An xsl style sheet containing the following styles has been saved to a text file named foo.xsl:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:output method="html" indent="yes"/>

  <xsl:template match="/">
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="s1">
    <html>
    <head><title><xsl:value-of select="@title"/></title></head>
    <body bgcolor="<@VAR request$bgcolour1>" text="#000000">
```

```

<xsl:apply-templates select="s2"/>
</body>
</html>
</xsl:template>

<xsl:template match="s2">
<table width="100%" border="0" cellspacing="0" cellpadding="4">
<tr>
    <td bgcolor="<@VAR request$bgcolour2">">
        <font color="#ffffff" size="+1">
            <b><xsl:value-of select="@title"/></b>
        </font>
    </td>
</tr>
</table>
<xsl:apply-templates/>
<br/>
</xsl:template>

<xsl:template match="p">
    <p><xsl:apply-templates/></p>
</xsl:template>
</xsl:stylesheet>

```

The following meta tag will apply the styleSheet to the DOM object MyDom.

```

<@XSLT object="MyDom" scope="request"
styleSheet="<@include file='<@appfilepath>xslt/
foo.xml'>">

```

If you wish to actually see the HTML which is the result of the transformation you will need to use HTML encoding as show below.

```

<pre><@XSLT object="request$MyDom"
styleSheet="<@include file='<@appfilepath>xslt/
foo.xml'>" ENCODING="HTML"><pre>

```

The following XML is output as a result of the XSLT transformation with HTML encoding set.

```

<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>sl foo</title>
</head>
<body bgcolor="" text="#000000">
<table width="100%" border="0" cellspacing="0" cellpadding="4">
<tr>
<td bgcolor=""><font color="#ffffff" size="+1"><b>Foo</b></font></td>
</tr>

```

```
</table>
<p>Hello</p>
<br>
<table width="100%" border="0" cellspacing="0" cellpadding="4">
<tr>
<td bgcolor=""><font color="#ffffff" size="+1"><b>Bar</b></font></td>
</tr>
</table>
<p>olleH</p>
<br>
</body>
</html>
```

<@SLEEP>

<@SLEEP>

Syntax

<@SLEEP VALUE="">

Description

Stops the current worker thread from processing for VALUE milliseconds. The thread that is processing the taf will go to sleep for this time without consuming system resources. This can be useful when you have cleared an error and need to wait a short period of time before retrying.

Example

To sleep the current worker thread for 500ms or half a second.

<@SLEEP VALUE="500">



Caution This meta tag does not affect the normal processing of other requests.

Modified Meta Tags

3

Meta Tags that have been changed or enhanced within Witango 5.5 functionality

This chapter outlines changes to syntax and functionality of the following existing Meta Tags:

<@CIPHER> on page 24

<@EMAIL> on page 27

<@CLEARERRORS> on page 28

<@TOKENIZE> on page 29

<@REPLACE> on page 30

<@SERVERSTATUS> on page 31

<@PURGECACHE> on page 32

<@CGIPARAM>, <@HTTPATTRIBUTE> on page 34

<@VAR> on page 37

<@CIPHER>

Changes

<@CIPHER> now supports strong encryption, HMAC, SHA, Hex and Base64 encoding.

Each type of cipher has at least one operation permitted. Each may accept a key, may provide a default key if none is given, or may reject any key and use a predetermined value, or none, as appropriate.

Cipher names are case insensitive.

Ciphers Supported

The following tables lists a short description of each cipher.

Type	Short Description
SHA	SHA256 Secure Hash Algorithm approved by the US Federation Information Processing Standards (FIPS) see http://csrc.nist.gov/cryptoToolKit .
MD5MAC	
HMAC_SHA	Symmetric Key algorithms used to create a Message Authentication Code when used with a specified hash algorithm.
TripleDES	Multiple of 8 bytes if encryption, text string if decryption. See http://csrc.nist.gov/CryptoToolKit
Rijndael	Multiple of 16 bytes if encryption, text string if decryption. Also known as AES. See http://csrc.nist.gov/CryptoToolKit
Blowfish	Multiple of 8 bytes if encryption, text string if decryption. See http://canterpane.com/blowfish.html
MARS	Multiple of 16 bytes if encryption, textstring if decryption. IBM Corporation 1994, 2003 See http://research.ibm.com/security/mars.html
Hex 2	Times of plain text length if encoding, text string if decoding. Hexadecimal Encoding Data.
Base64	Base 64 Encoding of Data. variable length if encoding text string if decoding.

Cipher Types

The following tables lists the new types of ciphers, their actions and their key restrictions.

Type	Action	Key Restrictions	Key Type
SHA	hash	ignored	n/a

Type	Action	Key Restrictions	Key Type
SHA256	hash	ignored	n/a
MD5MAC	hash	required ie, KEY=00112233445566778899aabbccddeeff" KEYTYPE="HEX" KEY=0123456789abcdef" KEYTYPE="TEXT"	Text/Hex Default:Text
HMAC_SHA	hash	required - variable length	Text/Hex Default isText
TripleDES	encrypt/ decrypt	required 16 byte if KeyType is Text 32 byte if KeyType is Hex	Text/Hex Default is Text
Rijndael	encrypt/ decrypt	required 16 byte if KeyType is Text 32 byte if KeyType is Hex	Cipher with variable block and key length. Text/Hex default is Text
Blowfish	encrypt/ decrypt	Required; 16 byte if KeyType is Text; 32 byte if KeyType is Hex	Symmetric Block cipher; Text/Hex; Default is Text
MARS	encrypt/ decrypt	Required; 16 byte if KeyType is Text; 32 byte if KeyType is Hex	Text/Hex; Default is Text
Hex	ncode/ decode	n/a	n/a
Base64	encode/ decode	n/a	n/a

Examples

```
<@CIPHER ACTION="encrypt" TYPE="Blowfish"
STR="BLOWFISH" KEY="abcdefghijklmnop" KEYTYPE="TEXT">
```

```
<@CIPHER ACTION="encrypt" TYPE="MARS"
STR="MARSEncryptionRocks" KEY="abcdefghijklmnop"
KEYTYPE="TEXT">
```

```
<@CIPHER ACTION="encrypt" TYPE="Rijndael"
STR="RijndaelisAlsoAES" KEY="abcdefghijklmnop"
KEYTYPE="TEXT">
```

```
<@CIPHER ACTION="encrypt" TYPE="TripleDES"
STR="TripleDESisTripleDES" KEY="abcdefghijklmnop"
KEYTYPE="TEXT">
```

<@CIPHER>

```
<@CIPHER ACTION="encode" TYPE="Hex"  
STR="HexEncodedString">
```

```
<@CIPHER ACTION="encode" TYPE="Base64"  
STR="Base64EncodedString">
```

<@EMAIL>

Changes

Whenever any data is returned from the mailserver as type=ARRAY using <@EMAIL>, Row 0 of the results array will now be populated with column names corresponding to the user selected field names.

These column names can be used to retrieve the data from the array.

Example

```
<@ASSIGN NAME="request$messageList" '<@EMAILSESSION  
LIST SESSIONID="POP3 Session: <@USERREFERENCE">'>
```

```
@request$messageList[1,ID]
```

will return the ID of the first message

```
@request$messageList[5,Size]
```

will return the size of the fifth message

<@CLEARERRORS>

<@CLEARERRORS>

Changes

In previous versions of the Witango Server the status code and reason phrase were left at 500 Internal Server Error. Now when <@CLEARERRORS> is called the http status code and reason phrase are reset back to 200 OK.

<@TOKENIZE>

Changes

The <@TOKENIZE> meta tag now accepts the parameter `STR=` as an alias of `VALUE=`. This has been a common support issue as it has been an inconsistency in the meta language.

The `VALUE` parameter will continue to be supported.

<@TOKENIZE> has also been extended to allow for two dimensional arrays to be created tokenising a string. Two new parameters have been added to allow for a `CDELIM` and `RDELIM` to be specified.

Example

The following tokenize action

```
<@TOKENIZE STR="this is text, and more.. and more"  
CDELIM="," RDELIM="." NULLTOKENS="true">
```

would return the array shown below:

this is text	and more
and more	

<@REPLACE>

Change

<@REPLACE> has been enhanced so regular expressions can be used in the `FINDSTR` attribute. This allows for more complex find and replace functions to be constructed. The tag now accepts `TYPE=Regex` parameter which flags the `FINDSTR` as a regular expression.

Example

```
<@REPLACE STR="abaaabaa" FINDSTR="a+" REPLACESTR="all"
TYPE="REGEX">
```

would produce the following string:

```
allballball
```

<@SERVERSTATUS>

Change

As part of the rewriting of the file caches and data source pool resources the following parameters have been added:

PARAMETER	DESCRIPTION
ActiveDataSrc	This parameter now returns an accurate number of data sources marked in use.
DataSrcCount	This parameter now returns an accurate number of data source connections allocated.
NumCachedDocs	This parameter will return the number of documents in the application file cache.
NumCachedIncl	This parameter will return the number of documents in the include file cache.
NumUsersLocal	This parameter now returns an accurate number of user references in the request variable store.
NumUsersShared	This parameter now returns an accurate number of user references in the shared variable store.

<@PURGECACHE>

Syntax

```
<@PURGECACHE [PATH=pathToPurge] [DOMAIN=ALL] [TYPES=
all| taf| include]>
```

Description

The purpose of the <@PURGECACHE> meta tag is to allow selective purging of Witango's file cache. The design of this meta tag includes considerations for Witango Servers deployed in an ISP environment.

The **PATH** attribute specifies the path (relative to the Web server's document root) to the directory (and any contained subdirectories) to purge. This path attribute only functions if the contents of `user$configPasswd` match `system$configPasswd`; that is, if the user has appropriate rights on the Witango system. For example, in an ISP environment, requiring that the password be set allows system administrators to purge the entire cache while restricting customers to purging only their own documents from the cache. The default value is the current path.

The **TYPE** attribute specifies the type of files to purge from the cache. `taf` refers to any application file; `include` refers to include files; `all` refers to both application and include files. The default value is `all`.

The **DOMAIN** attribute specifies the domain across which the purge cache will occur. If this attribute is set to `ALL`, the purge of document caches will occur across all domains. The default value is the current domain. For further information as to your current domain you should see <@DOMAIN> in the Witango Programmers Guide.

Examples

```
<@PURGECACHE>
```

Purges all files (tafs and included) cached from the calling application file's current path (this is equivalent to <@DOMAIN><@APPFILEPATH>), as well as any subdirectories.

```
<@PURGECACHE TYPES=include>
```

Purges all include files cached from the calling application file's current path (this is equivalent to <@DOMAIN><@APPFILEPATH>), as well as any subdirectories.

```
<@PURGECACHE PATH="/test">
```

If the variable `user$configPasswd` has not been set, this results in an error.

```
<@ASSIGN user$configPasswd value="myConfigPasswd">  
<@PURGECACHE PATH="/test">
```

Purges all files cached from the calling application file's `test` directory, as well as any subdirectories in the current domain.

```
<@ASSIGN user$configPasswd value="correctPassword">  
<@PURGECACHE DOMAIN="ALL">
```

Purges all files (tafs and included) cached from all directories and all domains on the current Witango Server. Effectively, this reinitialises the cache.



Note It is NOT recommended that an application cache be entirely purged on a regular basis as this will slow performance of your application. Attributes should be set according to the granularity of purging that is required.



Caution If the application you are migrating requires the purging of the entire cache of the Witango Server then you will need to modify your existing <@PURGECACHE> code to include `domain="all"`.

<@CGIPARAM>, <@HTTPATTRIBUTE>

Changes

As the web has evolved certain technologies have become less important. CGI is one of these technologies and as a consequence we are replacing <@CGIPARAM> with <@HTTPATTRIBUTE> which better describes the functionality of the tag.

The <@HTTPATTRIBUTE> tag has also been extended to expose:

- Content-Length
- SOAPAction
- Accept
- FullHeader

<@CGIPARAM> is now an alias of <@HTTPATTRIBUTE>

Syntax

<@HTTPATTRIBUTE NAME=*name* [ENCODING=*encoding*] >

Description

Evaluates to the value of the specified HTTP attribute. HTTP attributes are values passed to Witango Server by your Web server. HTTP attributes are passed whether you are using the CGI or the plug-in.

The following table shows valid values for the NAME attribute and descriptions of the value returned by each.

NAME attribute values	Description
ACCEPT	Specifies the media types which are acceptable for the response. If none are present then it is assumed that the client accepts all media types.
CLIENT_ADDRESS	The fully-qualified domain name of the user who called the application file, if your Web server is set to do DNS lookups; otherwise, this attribute contains the user's IP address. For example, "fred.xyz.com".
CLIENT_IP	The IP address of the user who called the application file. For example, "205.189.228.30".
CONTENT_LENGTH	Returns the length in bytes of the HTTP request.
CONTENT_TYPE	The MIME type of the HTTP request contents.
FULL_HEADER	Sends the full header to the Witango server.
FROM_USER	Rarely returns anything; with some older Web browser applications, the user's e-mail address.

NAME attribute values	Description
HTTP_COOKIE	Returns the value of the HTTP cookie specified in the COOKIE attribute. For example, <@HTTPATTRIBUTE NAME="HTTP_COOKIE" COOKIE="SICode"> returns the value of the SICode cookie. (This attribute is retained for backwards compatibility with Witango 2.3. It is recommended that you use <@VAR> with SCOPE="COOKIE" to return the values of cookies in Witango.)
HTTP_SEARCH_ARGS	Text after question mark (?) in the URL.
METHOD	The HTTP request method used for the current request. If a normal URL call, or form submitted with the GET method, "GET"; if a form submitted with the POST method, "POST".
PATH_ARGS	Text after the base URL (which includes the Witango CGI name, if present), and before any search arguments in the URL. <@APPFILE> returns the same value if there is no argument after the application file name and before any search arguments. For example, in the following two cases: (CGI) http://www.example.com/ Witango-bin/wcgi/fred search.taf?function=_form (plug-in) http://www.example.com/ fred/search.taf?function=_form <@HTTPATTRIBUTE NAME="PATH_ARGS"> returns: fred/search.taf
POST_ARGS	The raw POST (form submission) argument contents, containing the names and values of all form fields.
REFERER	The URL of the Web page from which the current request was initiated. Not provided by all Web browsers. (The misspelling of this attribute is for consistency with the CGI specification.)
SCRIPT_NAME	Returns the CGI portion of the URL.
SERVER_NAME	Fully-qualified domain name of the Web server, if your Web server is set to do DNS lookups; otherwise, this attribute contains the server's IP address. For example, "www.example.com".
SERVER_PORT	The TCP/IP port on which the Web server is running. A typical Web server runs on port 80.

<@CGIPARAM>, <@HTTPATTRIBUTE>

NAME attribute values	Description
SOAPAction	Returns the SOAPAction value it is specified in the HTTP header. This value is only relevant to the SOAP calls made to the Witango Server.
USERNAME	The user name, obtained with HTTP authentication, of the user who requested the URL. This attribute is available only if the URL used to call the current application file required authentication by the Web server software.
PASSWORD	The password, obtained with HTTP authentication, of the user who requested the URL. This attribute is available only if the URL used to call the current application file required authentication by the Web server software.
USER_AGENT	The internal name of the Web browser application being used to request the URL. This often contains information about the platform (Mac OS X, Windows, etc.) on which the Web browser is running, and the application's version.

Example

```
<P>Hi there, <TT><@HTTPATTRIBUTE  
NAME=CLIENT_ADDRESS>  
</TT>. You are connected to <TT><@HTTPATTRIBUTE  
NAME=SERVER_NAME></TT>, port <@HTTPATTRIBUTE  
NAME=SERVER_PORT>.
```

This returns a personalized greeting to the client, for example:

Hi there, whitman.leavesofgrass.com. You are connected to
baudelaire.flowersofevil.com, port 80.



Caution To bring your Witango application files up to the v5.5 meta language standard you should do a global find and replace of <@CGIPARAM> with <@HTTPATTRIBUTE>.

<@VAR>

Changes

The <@VAR> meta tag has been changed for DOM variables.

The <@VAR> tag will now accept XPATH and XPOINTER parameters to allow access to elements in DOM variables.

ELEMENT has been deprecated and is now aliased to XPOINTER.

Syntax

```
<@VAR NAME=name [SCOPE=scope] [XPOINTER=Xpointer]  
[TYPE=text] [ENCODING=encoding] [FORMAT=format] [XPATH=xpath]  
[{array attributes}]>
```

Shorthand for XPATH and XPOINTER

A shorthand representation has also been added for both XPATH and XPOINTER parameters to access variables. This short hand notation takes the form:

```
@@scope$VariableName[ {xptr|xpth}:"reference to element(s)" ]
```

Displaying HTML format in a DOM variable

If you display the content in HTML format in a DOM variable you can now do so with the attribute ENCODING="HTML".

Before you had to assign the XML text to a variable and then return the value of that variable:

```
<@ASSIGN tempXML <@VAR myDOM TYPE=TEXT>>  
<@VAR tempXML>
```

Now:

```
<@VAR myDOM TYPE=TEXT Encoding="HTML">
```

<@DOM>, <@DOMINSERT>, <@DOMREPLACE>, <@DOMDELETE>, <@ELEMENTATTRIBUTE>, <@ELEMENTATTRIBUTES>, <@ELEMENTNAME>, <@ELEMENTVALUE>

Change

All the XML related tags have been extended accept XPath as a means of consuming a DOM. All these tags now support DOM2, XPointer and Xpath.



Note In versions prior to Witango 5.5 the attribute used to access a DOM was known as `ELEMENT`. This attribute name has now been deprecated and is aliased to `XPOINTER`.

Example

```
<XML>

  <DIV>

    <P>Paragraph 1</P>
    <P>Paragraph 2</P>

  </DIV>
  <DIV>

    <P>Paragraph 3</P>
    <P>Paragraph 4</P>

  </DIV>
</XML>

<@assign name="mydom"
  value="<@DOM VALUE='
  <XML>

    <DIV>

      <P>Paragraph 1</P>
      <P>Paragraph 2</P>

    </DIV>
    <DIV>

      <P>Paragraph 3</P>
      <P>Paragraph 4</P>

    </DIV>
  </XML>
```

<@DOM>, <@DOMINSERT>, <@DOMREPLACE>, <@DOMDELETE>, <@ELEMENTATTRIBUTE>, <@ELEMENTAT-

'>
>

XPath

```
<@DOMINSERT object=MyDom scope=Request xpath="/  
child::root/child::*[1]" position=append>
```

```
<@DOMREPLACE object=MyDom XPATH="//P[@attr='attr value  
of p3']">
```

```
<@DOMDELETE object=MyDom XPath="//DIV/child::LI">
```

```
<@ELEMENTATTRIBUTE object=MyDom Xpath="/child::root/  
*[1]/*[3]/*" attribute=attr>
```

```
<@ELEMENTNAME object=MyDom xpath="/child::root/  
child::*" type="array">
```

```
<@ELEMENTVALUE object=MyDom xpath="/child::root/  
child::*[1]/child::*[2]" type="array">
```

XPointer

```
<@DOMINSERT object=MyDom scope=Request  
xpointer="child(1)" position=append>
```

```
<@DOMREPLACE object=MyDom  
XPOINTER="root.descendant(all,P,attr,attr value of  
p3)">
```

```
<@DOMDELETE object=MyDom  
xpointer="child(1).child(all,LI)">
```

```
<@ELEMENTATTRIBUTE object=MyDom  
xpointer="child(1).child(3).child(all)" attribute=attr>
```

```
<@ELEMENTNAME object=MyDom xpointer="child(all)">
```

```
<@ELEMENTVALUE object=MyDom  
xpointer="child(1).child(2)" type="array">
```

<@DOM>, <@DOMINSERT>, <@DOMREPLACE>, <@DOMDELETE>, <@ELEMENTATTRIBUTE>, <@ELEMENTAT-

New Features

4

The following functionality has been introduced or enhanced in the Witango Server 5.5

This chapter outlines the following feature changes and enhancements to the Witango Server 5.5:

XPATH on page 42

Web Services on page 45

Server Configuration Changes on page 64

Changes to Encoding on page 67

Licensing File Changes on page 69

XPATH

Changes

All the XML related tags in Witango (`<@DOM>`, `<@DOMINSERT>`, `<@DOMREPLACE>`, `<@DOMDELETE>`, `<@ELEMENTATTRIBUTE>`, `<@ELEMENTNAME>`, `<@ELEMENTVALUE>`) have been extended accept XPath as a means of consuming a DOM. All these tags now support DOM2, XPointer and Xpath. XPath and XPointer are specified by the World-Wide Web Consortium.



Note For detailed information on XPath, see the W3C website at www.w3.org/TR/xpath.



Note For detail information on XPointer, see the W3C website at www.w3.org/TR/xptr-framework.

XPath Syntax

XPath is used by the various `<@DOM. . . >` and `<@ELEMENT. . . >` , metatags to point at one or more elements in a DOM using path expressions. These path expressions look very much like the expressions you see when you work with a computer file system.

XPath uses a pattern expression to identify nodes in an XML tree. It is a syntax for “addressing” (accessing) specific parts of XML data. The expression is used to select elements from the XML tree which match the expression. XPath is always read left to right.

The XPath pattern is a slash (/) separated list of child element names that describe a path or expression through the XML document. This syntax has support for:

- string functions and formatting
- math and number recognition
- boolean values to create complex expressions.

If the path starts with a slash (/) it represents an absolute path to an element. If the path starts with two slashes (//) then all elements in the document that fulfill the criteria will be selected even if they are at different levels in the xml tree.

By using square brackets in an XPath expression you can specify an element further and wildcards (*) can be used to select unknown XML element(s) at any location in an XPath expression.

XPath Functions

XPath expressions support the following functions:

Boolean
boolean()
true()
false()
lang()
not()

Node-Set
count()
id()
last()
local-name()
name()
namespace-uri()

String
concat()
contains()
normalize-space
starts-with()
string()
string-length()
substring()
substring-after()
substring-before()
translate()

Number
ceiling()
floor()
number()
round()
sum()

Example

Given the following document instance:

```
<catalogues>
  <catalogue type="CDs">
    <cd id="1">
      <title>Back in Black</title>
```

```

        <artist country="AU">AC/DC</artist>
        <price currency="AU">19.99</price>
        <stocklevel>1</stocklevel>
    </cd>
    <cd id="2" country="AU">
        <title>10 9 8 7 6 5 4 3 2 1</title>
        <artist country="AU">Midnight Oil</artist>
        <price currency="AU">14.99</price>
        <stocklevel>10</stocklevel>
    </cd>
    <cd id="3" country="UK">
        <title>Dark Side of the Moon</title>
        <artist country="UK">Pink Floyd</artist>
        <price currency="AU">14.99</price>
        <stocklevel>0</stocklevel>
    </cd>
</catalogue>
<catalogue type="Books">
</catalogue>
<catalogue type="DVDs">
</catalogue>
</catalogues>

```

* /*

returns the catalogues element as would

/catalogues

* /catalogues/catalogue[@type='CDs']/cd/*

returns all elements within the cd elements in the catalogue node where type equals cd

* /catalogues/catalogue/cd[@country='UK']/title

will return the title element of the cd element where country equals UK

Web Services

The Witango Server 5.5 is capable of accepting SOAP document literal web service requests.

This is achieved by exposing `tcf` files as SOAP document literal web services. In effect, any `tcf` file that has been published as a web service through the appropriate server settings will respond with a SOAP response without the user having to write a single line of code.

A `tcf` file is published as a web service by:

- Enabling the web services functions on the Witango Server so that a SOAP request can be received and processed.
- Register a file extension (we use `.wws`) with your Web Server so that these web service requests will be passed to the Witango Server Plugin.
- Creating a WSDL file for your `tcf` (The WSDL file outlines which methods of the `tcf` are exposed through the web service). A tool is available on the `witango.com` website to create your WSDL file.
- Configuring the Witango Server so that it knows which WSDL files are linked to which `tcf` files.



Note Witango's SOAP document literal web services are compatible with Microsoft .Net web services and are easily integrated into a .Net architecture.

Web Services Overview

There are two competing standards for the implementation of a web service:

- XML-RPC (XML Remote Procedure Call)
- SOAP (Simple Object Access Protocol)

Witango provides support for the publishing of a web service based on a SOAP implementation.

Witango provides support for the consumption of a web services based on either SOAP or XML-RPC.

What is a web service?

A web service is an application:

- that is capable of being defined;

- that can be located via the internet protocol;
- that is capable of interacting with other software applications; and
- that can be identified by a Uniform Resource Identity.

To exploit these applications, a web service consumer must be able to bind to a service and access the functions via the published interface. This is achieved through a number of fundamental building blocks outlined below.

What is a web service made of?

The fundamental building blocks of a web service are:

XML - the language used in the communication - it is a universal way of describing structured documents and data;

SOAP (Simple Object Access Protocol) is the messaging protocol via which web services communicate- it defines a uniform way of moving messages between services described by WSDL interfaces.

HTTP - the transport protocol for the communication;

WSDL (Web Services Description Language) is the technology used by a web service to publish its interfaces to the network. It is used to define the location of the web service, the functions it implements and how to access and use each function.

UDDI (Universal Description, Discovery and Integration) which is a registry and a protocol for publishing and discovering web services. It is a directory of web services that are available from different companies.

What is a SOAP message made up of?

A SOAP message is an XML document with a root element known as an *envelope*. Within the envelope there are two child elements:

- a *header* element, which is optional and is used to extend messages for circumstances such as authentication or transaction support;
- a *body* element, which is mandatory and is used to carry the actual SOAP message, also known as the “payload”.

The SOAP envelope defines the overall framework for expressing:

- what is in a message;
- who should deal with it; and
- whether it is optional or mandatory.

What is a WSDL document ?

A WSDL document is a complex mechanism for defining and describing interfaces to a Web Service. It defines what the service can do, where it can be found and how it can be invoked.

A WSDL document is an XML document with a specific format that describes:

- concrete parts of the web service such as services and bindings; and
- abstract part of the web service such as ports, messages, types of messages.

It is a very complex structure. Witango has a WSDL generator available on witango.com which can be used to generate this file automatically without any knowledge of its contents. For more information, see “Creating a WSDL file for your tcf” on page 49.

How do you send a SOAP message?

SOAP messages can go over any number of different protocols - but most often HTTP is used, with a SOAP message being sent as a HTTP POST request.

A HTTP POST request is structured such that:

- The first line of the request contains the method, the URI of the recipient and the HTTP version.
- The second line contains the IP address of the sender.
- The third line specifies the MIME type and the character encoding of the request.
- The fourth line tells the server how many characters to expect in the payload.
- Then there is an extra carriage return/linefeed.
- Then the payload itself.

eg:

```
M-POST /path HTTP 1.1
Host: <ip address of sender>
Content-Type: text/xml; charset="utf-8"
Content-Length: <length of payload>

<payload>
```

Enabling Web Services

Before publishing a TCF as a SOAP Web Service you must start the Web Service functions the Witango Server and associate a file extension with your web services. This is done in the `witango.ini` file with 2 new parameters `WEBSERVICES` and `WEBSERVICESEXTNS`.

WEBSERVICES can be either `TRUE` or `FALSE`. If the parameter is set to `TRUE` the Witango Server will treat any request with a file extension of `WEBSERVICESEXTNS` as a SOAP request and respond to the request with a SOAP response.

WEBSERVICESEXTNS is a semi colon (;) separated list of extensions that you will use to denote that a request contains a SOAP payload.

Sample web service setup

In the example below Web Services have been turned on and the file extension `wws` has been associated with a web service call (SOAP request).

```
witango.ini  
  
WEBSERVICES=true  
WEBSERVICESEXTNS=wws
```

Once the web services have been enabled in the `witango.ini` file and the Witango Server has been restarted you will notice the following line in the `witangoevents.log` file:

```
Web Service: Enabled
```

This status line indicates whether web services have been turned on.



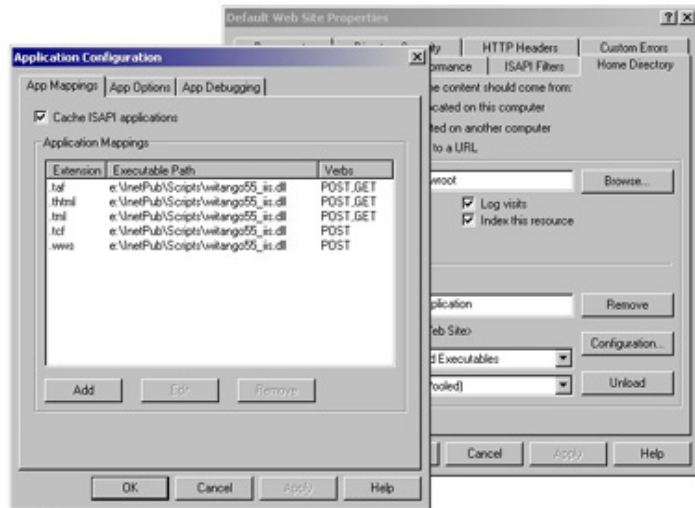
Note These Web Services parameters can be controlled using the Witango 5.5 Administration Application.

Register a file extension

A file extension (we use `.wws`) needs to be registered with your Web Server so that web service requests will be passed to the Witango Server Plugin.

How this is done will depend on the Web Server you use.

Example for IIS:



See the documentation for your web server on the procedure to add file mappings to your web server's configuration.

Creating a WSDL file for your tcf

A utility has been written that will automatically create a WSDL file from a tcf. This utility is available in the Developer section of the witango.com web site.



Note It is not recommended that you try to construct a WSDL file manually without a comprehensive knowledge of the structure of WSDL and SOAP requests.

Steps to Create your WSDL file

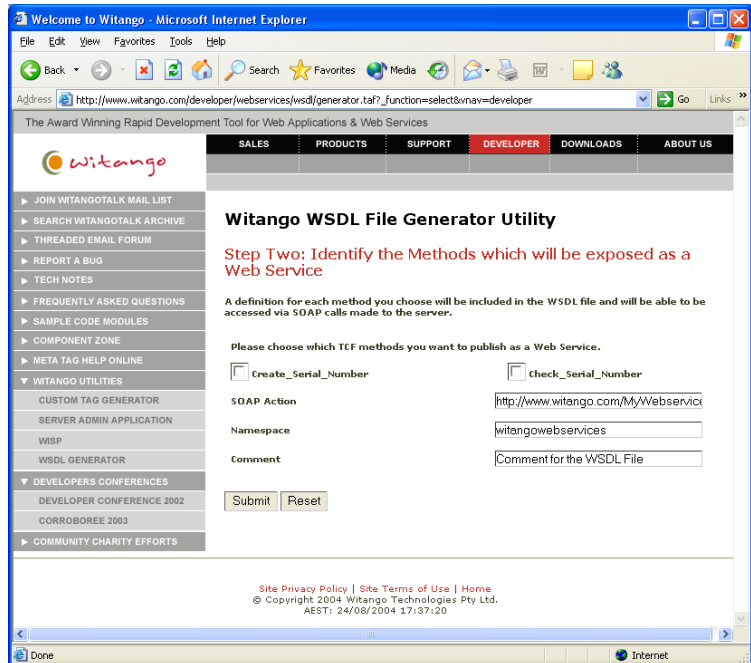
- I Go to the Witango WSDL File Generator Utility at <http://www.witango.com>

2 The Form shown below will appear.

The screenshot shows a web browser window titled "Welcome to Witango - Microsoft Internet Explorer". The address bar displays the URL: `http://www.witango.com/developer/webservices/wsd/generator.taf?vnav=developer`. The page features a navigation menu with links: SALES, PRODUCTS, SUPPORT, DEVELOPER (highlighted), DOWNLOADS, and ABOUT US. On the left, a sidebar lists various resources, with "WSDL Generator" highlighted under the "DEVELOPERS CONFERENCE" section. The main content area is titled "Witango WSDL File Generator Utility" and contains the following text: "Step One: Identify the TCF File" and "Please select a TCF file to create a WSDL file for.". Below this text is a form with a label "TCF Files:" followed by a text input field and a "Browse..." button. At the bottom of the form are "Submit" and "reset" buttons. The footer of the page includes links for "Site Privacy Policy", "Site Terms of Use", and "Home", along with copyright information: "© Copyright 2004 Witango Technologies Pty Ltd. AEST: 24/08/2004 17:34:07".

3 Select the BROWSE button to locate your tcf file and then press the SUBMIT button.

- 4 The next screen will appear allowing you to specify which methods within the chosen tcf file you wish to expose in your Web Service together with other configuration settings.



- 5 Check the boxes for the methods you wish to expose in this web service.
- 6 Set the **SOAP Action** allows the user to indicate the intent of the SOAP HTTP request. The value is a URI identifying the intent. No value means that there is no indication of the intent of the message. Basic Witango Web Services do not require this field - so it can be left as an empty string.
- 7 Set the **Namespace** allows the user to provide a namespace to avoid a conflict with element names.
- 8 Set the **Comment** this is optional and can be used to add a comment to the WSDL file.

- 9 Click **Submit** button, the contents of your WSDL file will be rendered to screen in a FORM field.

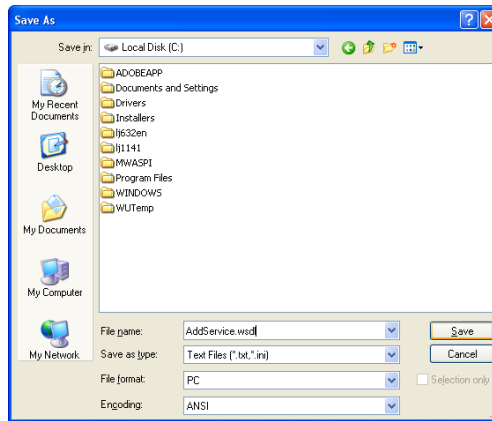


Note The contents of the WSDL file is typically very large and you will need to scroll through the entire field to see it all.

```
<definitions name="add_Service"
targetNamespace="vitangovebservices"
xmlns:tns="vitangovebservices"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns="http://schemas.xmlsoap.org/wsdl/"
>
<types>
  <xsd:schema targetNamespace="vitangovebservices" xmlns="vitangovebservices">
    <xsd:complexType name="ArrayOfString">
      <xsd:sequence>
        <xsd:element minOccurs="0" maxOccurs="unbounded" name="vitangostring" nillable="true"
type="xsd:string" />
      </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="Add_Out_Request">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element minOccurs="0" maxOccurs="1" name="Value1" nillable="true"
type="xsd:string" />
          <xsd:element minOccurs="0" maxOccurs="1" name="Value2" nillable="true"
type="xsd:string" />
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="Add_Out_Response">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element minOccurs="0" maxOccurs="1" name="ParamOutResult"
nillable="true" type="xsd:string" />
          <xsd:element minOccurs="1" maxOccurs="1"
name="Add_Out_HTMLResult" nillable="true" type="xsd:string" elementType="MethodResult" />
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="Add_Out_Fault">
      <xsd:complexType>
        <xsd:all>
          <xsd:element name="errorMessage" type="xsd:string"/>
        </xsd:all>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="Add_In_Request">
      <xsd:complexType>
```

- 10 Highlight ALL of the text and **COPY** it to your clipboard.

I2 Save the new document as a <webservicename>.wsdl file.



I3 Move the WSDL file to your Witango Server.



Note You can place this file where ever you require on the Witango Server but the standard location would be within your web site

Configuring the Witango Server for Web Services

Each Web Service must be configured in the the `webservices.ini` file.

To configure a web service you will need to create the appropriate stanza in the `webservices.ini` file which is located in the configuration directory. The purpose of each stanza is to name the web service and link it to the corresponding `tcf` and `WSDL` files.

It is the `stanzaName` in conjunction with the `WEBSERVICESEXTNS` settings that defines whether a SOAP call is valid.

The `webservices.ini` file has the same format as the other Witango configuration files, which is as follows:

```
stanzaName= comment
[stanzaName]
parameterName=parameterValue
```

Sample web service configuration

```
webservices.ini
SampleService=
```

```
[SampleService]
webservice=webservice/SampleService.tcf
wsdlfile=webservice/SampleService.wsdl
```

eg

```
http://127.0.0.1/SampleService.wws
```

This does not map to a physical file but is mapped to a `tcf` and an associated WSDL file.

SOAP related variables and parameters

```
<@HTTPATTRIBUTE SOAPAction>
```

Returns the value of the SOAP action from the HTTP header.

```
@@request$SOAPHeader
```

Is automatically populated with the header of the SOAP request if it is present. This is a DOM variable.

```
@@request$SOAPBody
```

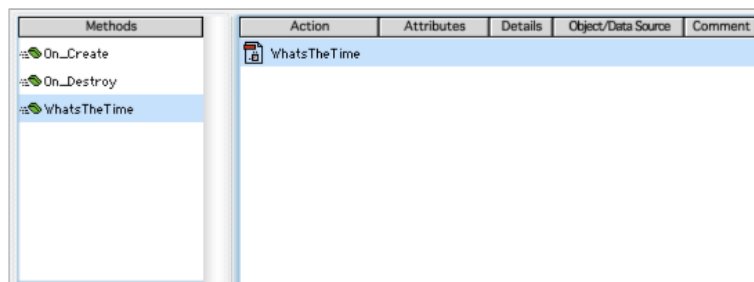
Is automatically populated with the body of the SOAP request.

How to consume your Witango Web Service

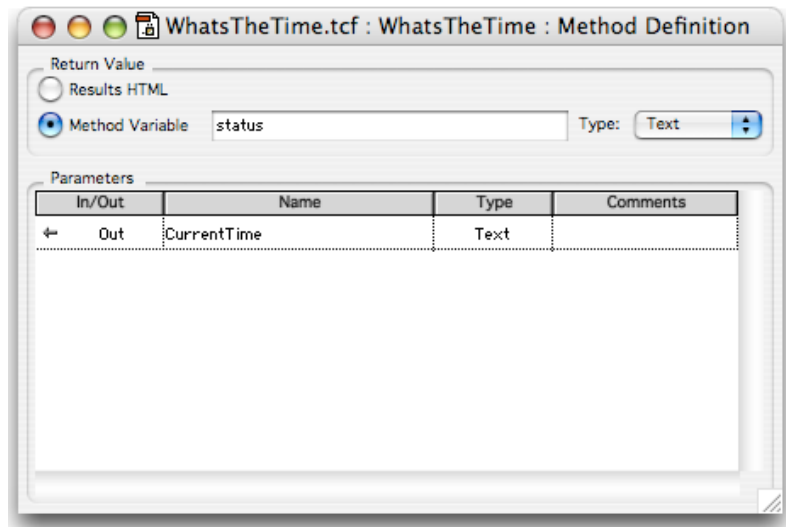
Creating a Simple Web Service

Lets start by creating a simple web service that returns the current time on the server.

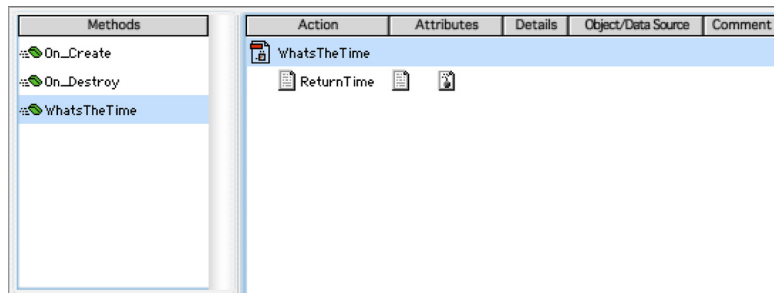
- 1 Create new `tcf` and save it as `WhatsTheTime.tcf` in the `<webroot>/webservices/WhatsTheTime` directory
- 2 Add a new method and call it `WhatsTheTime`



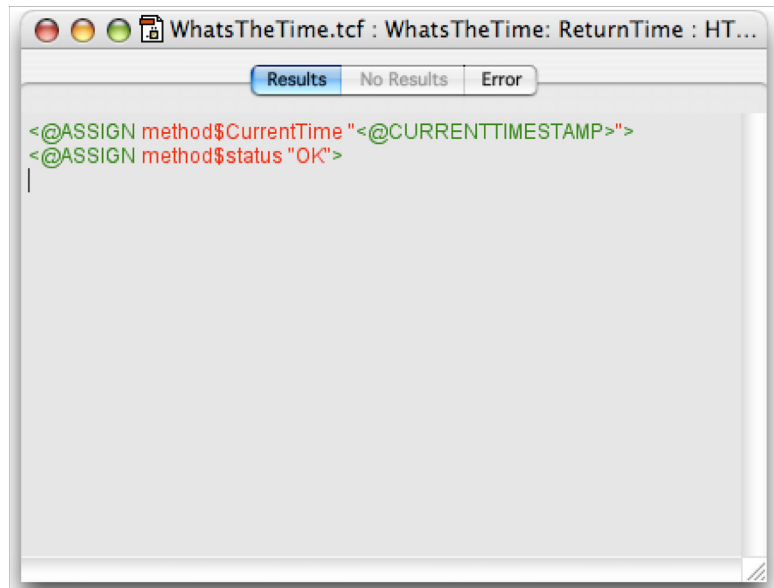
- 3 Add an Out parameter named `currentTime` with a type of `Text`. Change the Return Value to be Method Variable named `status`.



- 4 Add a Result action to the method and name it `ReturnTime`.

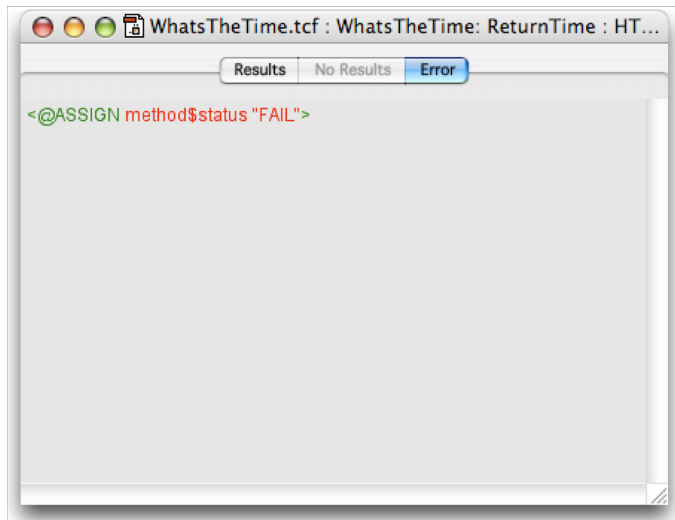


5 In the Results tab add the following code:



```
<@ASSIGN method$CurrentTime "<@CURRENTTIMESTAMP>">  
<@ASSIGN method$status "OK">
```

6 In the Error tab add the following code:



For more information, see
“Creating a WSDL file for
your tcf” on page 49

7 Save the file.

- 8 Create a WSDL file for the WhatsTheTime.tcf file and save it as WhatsTheTime.wsdl in the same directory as the tcf. It will look similar to the following wsdl structure. Your file may have a different location and SOAPAction value. For this example we are using localhost.**

```
<definitions name="WhatsTheTime_Service"
targetNamespace="witangowebsservices"
xmlns:tns="witangowebsservices"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns="http://schemas.xmlsoap.org/wsdl/"
>
<types>
<xsd:schema targetNamespace="witangowebsservices"
xmlns="witangowebsservices">
  <xsd:complexType name="ArrayOfString">
    <xsd:sequence>
      <xsd:element minOccurs="0"
maxOccurs="unbounded" name="witangostring" nillable="true"
type="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="WhatsTheTime_Request">
    <xsd:complexType>
      <xsd:sequence></xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="WhatsTheTime_Response">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element minOccurs="0"
maxOccurs="1" name="CurrentTime" nillable="true"
type="xsd:string" />
        <xsd:element minOccurs="0"
maxOccurs="1" name="status" nillable="true"
type="xsd:string" elementType="MethodResult" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="WhatsTheTime_Fault">
    <xsd:complexType>
      <xsd:all>
        <xsd:element name="errorMessage"
type="xsd:string"/>
      </xsd:all>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
</types>
<message name="WhatsTheTime_Request_Msg">
  <part name="WhatsTheTime_Request_Part"
element="tns:WhatsTheTime_Request"/>
</message>
```

```

</message>
<message name="WhatsTheTime_Response_Msg">
  <part name="WhatsTheTime_Response_Part"
    element="tns:WhatsTheTime_Response"/>
</message>
<message name="WhatsTheTime_Fault_Msg">
  <part name="WhatsTheTime_Fault_Part"
    element="tns:WhatsTheTime_Fault"/>
</message>
<portType name="WhatsTheTimePortType">
  <operation name="WhatsTheTime">
    <input message="tns:WhatsTheTime_Request_Msg"/>
    <output
      message="tns:WhatsTheTime_Response_Msg"/>
    <fault message="tns:WhatsTheTime_Fault_Msg"/>
  </operation>
</portType>
<binding name="WhatsTheTimeBinding"
  type="tns:WhatsTheTimePortType">
  <soap:binding style="document" transport="http://
schemas.xmlsoap.org/soap/http"/>
  <operation name="WhatsTheTime">
    <soap:operation soapAction="http://localhost/
WhatsTheTime.wws" style="document"/>
    <input><soap:body use="literal"
      namespace="witangowebsservices"/></input>
    <output><soap:body use="literal"
      namespace="witangowebsservices"/></output>
    <fault><soap:body use="literal"
      namespace="witangowebsservices"/></fault>
  </operation>
</binding>
<service name="WhatsTheTime">
  <documentation>Service to provide the time on a remote
  server</documentation>
  <port name="WhatsTheTimePort"
    binding="tns:WhatsTheTimeBinding">
    <soap:address location="http://localhost/
WhatsTheTime.wws"/>
  </port>
</service>
</definitions>

```

For more information, see
“Enabling Web Services” on
page 48.

- 9 Once you have created a WSDL file you can configure the `webservices.ini` file so your web service will be registered with the Witango Server.

```

[webservices]
WhatsTheTime=

[WhatsTheTime]
webservice=webservices/WhatsTheTime/WhatsTheTime.tcf
wsdlfile=webservices/WhatsTheTime/WhatsTheTime.wsdl

```

- 10** Restart the Witango Server and check the `witangoevents.log` file to ensure that Web Services are enabled and your `WhatsTheTime` web service has been registered.

```
START    INFO    Web Service: Enabled
START    INFO    Web Service: Registering WhatsTheTime
```

Calling the Web Service via HTTP

To illustrate the process of a SOAP communication, the example below will show the packet data transmitted from client to server and vice versa.

To build a client from scratch you must:

- first get the WSDL file with the relevant web service definitions for the remote server;
- identify the information defining the message you wish to call;
- then create a set of parameters to pass into the Web service and then remotely call the `WhatsTheTime` method.

Sending a request to the web service to return the current date and time

- 1** Request the WSDL file from the server. This file will describe what interfaces can be called via SOAP calls. This is done with a http request for the file. In your browser send a request to:

```
http://localhost/webservices/WhatsTheTime/
WhatsTheTime.wsdl

GET /webservices/WhatsTheTime/WhatsTheTime.wsdl HTTP/1.1
Host: localhost:80
```

View the source and you will see the XML that makes up the WSDL describing our web service. It contains definitions for all the methods available for our `WhatsTheTime` web service.

Before you can construct the SOAP payload you need to identify:

- the method to call (operation input message),
 - the message that corresponds to the method call (message),
 - the parameters that will be passed in the message (associated elements), and
 - the response that will be received back (operation output message).
- 2** In this case there is a single method or operation with no parameters called `WhatsTheTime`.

```

    <operation name="WhatsTheTime">
        <input message="tns:WhatsTheTime_Request_Msg"
        <output message="tns:WhatsTheTime_Response_Msg"/>
        <fault message="tns:WhatsTheTime_Fault_Msg"/>
    </operation>

```

- 3 Use the input of the operation which will send the WhatsTheTime_Request_Msg message. This message is a method call with no parameters (elements) being passed:**

```

<message name="WhatsTheTime_Request_Msg">
<part name="WhatsTheTime_Request_Part"
element="tns:WhatsTheTime_Request"/>
</message>

```

```

<xsd:element name="WhatsTheTime_Request">
<xsd:complexType>
    <xsd:sequence></xsd:sequence>
</xsd:complexType>
</xsd:element>

```

- 4 From this information you can construct the SOAP envelope. The envelope must adhere to the SOAP protocol rules:**

```

<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/
envelope">
<env:Body>
    <mns:WhatsTheTime_Request
xmlns:mns="witangowebsservices" />
</env:Body>
</env:Envelope>

```

- 5 The parameters are sent according to the Web service specifications. To send the SOAP envelope to the server you need to POST the payload to the server using HTTP:**

```

Host: localhost:80
Accept: */*
User-Agent: Witango Server 5.5
Content-Type: text/xml; charset="UTF-8"
Content-Length: 245

SoapAction: http://localhost:80/WhatsTheTime.wws

```

```

<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/
envelope">
    <env:Body>
        <mns:WhatsTheTime_Request
xmlns:mns="witangowebsservices" />
    </env:Body>
</env:Envelope>

```

- 6 The web service will respond with the result of our query wrapped as a SOAP message in the format described in the output message of the WSDL file.

```

STATUS: 200 OK
Date: Wed, 12 Aug 2004 01:46:49 GMT
Server: Apache/2.0.48 (Unix) DAV/2
Set-Cookie:
Witango_UserReference=AD09F1C0026A778E412BEF89; path=/
Keep-Alive: timeout=15, max=92
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=ISO-8859-1

<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/
envelope">
<env:Body>
    <mns:WhatsTheTime_Response
xmlns:mns="witangoweb services">
        <CurrentTime>12/08/2004 11:46:49</
CurrentTime>
        <status></status>
    </mns:WhatsTheTime_Response>
</env:Body>
</env:Envelope>

```

Calling the web service via a Witango File

Applying the above example to a Witango script you can achieve this call by the following meta tags which define the location (WebService), SOAPAction and SOAP envelope (SoapPayload) for the web service. These are then used with in an @URL tag which performs a http POST to the server. The response is then stored in a DOM variable named SOAPResponse.

```

<@ASSIGN request$WebService "http://localhost/
WhatsTheTime.wws">

<@ASSIGN request$SOAPAction "http://localhost/
WhatsTheTime.wws">

<@ASSIGN request$SoapPayload '

<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/
envelope">

<env:Body>

    <mns:WhatsTheTime_Request
xmlns:mns="witangoweb services" />

</env:Body>

</env:Envelope>

```

```
'>

<@ASSIGN Request$SOAPResponse '<@DOM VALUE="

<@URL

LOCATION='@@request$WebService'

USERAGENT='Witango <@VERSION> <@PLATFORM><@CRLF>Content-
Type: text/xml<@CRLF>SOAPAction: "@@request$SOAPAction"'

POSTARGS='<?xml version="1.0" encoding="ISO-8859-1"
?><@CRLF><@VAR request$SoapPayload ENCODING="NONE">'

>">'>

>
```

The results of the SOAP request can be displayed to the browser by using a html encode on the variable and its format can be maintained by wrapping it in `<pre>` tags.

```
<pre>
<@VAR Request$SOAPResponse encoding="html">
</pre>
```

You would not usually write the output of a SOAP response to screen in its raw state. You would normally perform some kind of transformation over the DOM or reference some of the elements to be used in your application. With this simple web service you could check the status was OK and if it was use the timestamp.

```
<@IF "'<@ELEMENTVALUE Request$SOAPResponse XPATH="//*/
status">'='OK'">

<@ELEMENTVALUE Request$SOAPResponse XPATH="//*/
currentTime">

</@IF>
```

Server Configuration Changes

Client.ini

Two new parameters have been added to the `client.ini` file for configuring web server plug. They are configured in each of the the plugin stanza.

```
SENDERFULLHEADER=[TRUE| FALSE]
```

```
REGISTER_EXTNS=tcf;xmlx;wvs;taf
```

REGISTER_EXTNS is not valid for the IIS plug-in as IIS does not allow the plug-in to register extensions. To register extensions for IIS you need to use the Internet Services Manager.



Note Enabling `SENDERFULLHEADER` will increase the network traffic between the plug and the Witango Server and may impact on the performance of the web server, network and Witango server.

Witango.ini

`CONFIGPASSWD` is now case sensitive.

`WEBSERVICES` can be either `TRUE` or `FALSE`. If the parameter is set to `TRUE` the Witango Server will treat any request with a file extension of `WEBSERVICESEXTNS` as a SOAP request and respond to the request with a SOAP response.

`WEBSERVICESEXTNS` is a semi colon (;) separated list of extensions that you will use to denote that a request contains a SOAP payload.

The `SYSTEMCONFIGURATION` password is blocked.

Web Services can now be enabled on the server and an extension associated with web service calls (SOAP) to the server.

`ENCODEHTTPRESPONSE` has replaced `ENCODERESULTS` which has been removed.

`ENCODEHTTPRESPONSE` better describes what is being encoded and when the server applies the encoding. This parameter controls whether high ascii characters are encoded their html representations (e.g. `#174;`). The parameter can be set to `true` or `false`. If `ENCODEHTTPRESPONSE` is set to `true` the http response to the browser will have high ascii characters encoded. If you have overridden the setting of `ENCODERESULTS` in other scopes in your application files you will need to replace each instance of its use with `ENCODEHTTPRESPONSE`.

`ENCODERESULTHTML` has been added to allow the programmer or system administrator control of the html encoding that occurs as results

are pushed onto the results buffer during TAF execution. The parameter can be set to true or false. The default value is false. This encoding converts the following characters:

<	<
>	>
"	"
&	&

The `LISTENERPORT` parameter is now set to 18155 to prevent conflicts with Witango Server 5.0 which uses port 18100. This means that both 5.0 and 5.5 servers can be running at the same time on a single server during the upgrade phase of your service.

`SENDUSERREFERENCECOOKIE` has been added to control the sending of the Witango user reference cookie to control session management. The parameter takes a value of either TRUE or FALSE. When set to false, the session management must be controlled via search args or post args sending the userreference. This system variable can be set in all scopes.

`LOGARGUMENTS` has been added as a security enhancement to stop sensitive information being written to the `Witango.log`.

`LOGARGUMENTS` determines whether post and searchargs are written to the log files. This parameter overrides all logging levels.

`LOGARGUMENTS` can be set to TRUE or FALSE. The default value is FALSE.

Data Source Pool

The resources that control the connection management and sharing of data sources has been rewritten to provide faster access to the data source pool and increase stability under heavy load. The data source pool is now broken into a 2 level cache based on data source type and name. It also has a more efficient algorithm to find an available connection.

Document and Include cache

A new mechanism has been implemented to replace the old document cache. The new cache is more efficient under load and separates the include files and application files into different resources. This lowers the contention on the cache providing greater efficiency and stability under load. The new cache also provides more accurate reporting.

DOM variables

DOM variables are now based on the DOM2 standard.

Existing DOMS may not work as in `value="<"` is no longer permitted and need to add `<?xml version...>`

Witango Event log

The `witangoevents` log has been updated to report the license information, custom tag that are registered during startup and the web service that are being published.

JavaScript Engine

The javascript library has been updated to use the Mozilla Spider Monkey library. It is now thread safe.

Improved error messages in web server plugins.

The reporting of errors while initialising the web server plugin has been improved.

Web Server Module Name Changes

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

Module	Witango 5.0	Non-Windows Name	Windows Name
Apache 1.3 plug-in	wapache.so	witango55_apache.so	witango55_apache.dll
Apache 2.0 plug-in	wapache2.s0	witango55_wapache2.so	witango55_apache2.dll
IIS plug-in	wiis.dll	-	wiis.dll
CGI	wcgi.exe	witango55_cgi	wcgi.exe
SUN	-	witango55_sun.exe	witango55_sun.dll

Changes to Encoding

Changes

There have been major changes to how the encoding in the Witango Server works.

The default encoding is now `NONE` (previous versions this was `HTML`) and the encoding methods have been expanded and cleaned up.

The **default encoding method** for the server is now `NONE`. The previous setting was `HTML` in a result actions and `NONE` in other actions. For consistency the default is `NONE` for all actions and meta tags. A new encoding method `HTML` has been added to force a `HTML` encoding of a string.

Encoding has now been enhanced to include `NONE`, `HTML`, `MULTILINE`, `MULTILINEHTML`, `META`, `METAHTML`, `URL`, `JAVASCRIPT`, `SQL` and `CDATA`. These encodings have been standardized to describe what encoding is being applied to the string.

This has changed the functionality of `METAHTML`, `MULTILINE` and `MULTILINEHTML`.

In previous versions of Witango (and Tango) `METAHTML` would evaluate metatags but would not do `HTML` encoding. You should now replace `METAHTML` in `taf` files with `ENCODING="META"`.

In previous versions of Witango (and Tango) `MULTILINEHTML` would replace `cr/lf` with `<br>` but would not do `HTML` encoding. You should replace `MULTILINEHTML` with `MULTILINE`.

In previous versions of Witango (and Tango) `MULTILINE` would replace `cr/lf` with `<br>` AND do `HTML` encoding. You should replace `MULTILINE` with `MULTILINEHTML`.

These changes are summarised in the table below:

ENCODING	Witango 5.0	Witango 5.5	Change Required
HTML	-	HTML encoding	No
NONE	-	-	No
MULTILINE	HTML encoding /r to 	No HTML encoding /r to 	Yes
MULTILINEHTML	No HTML encoding /r to 	HTML encoding /r to 	Yes

ENCODING	Witango 5.0	Witango 5.5	Change Required
META	-	No HTML encoding Evaluate Meta Tag	No
METAHTML	No HTML encoding Evaluate Meta Tag	HTML encoding Evaluate Meta Tag	Yes



Conversion to 5.5

Caution Do NOT immediately replace MULTILINE and MULTILINEHTML as you may lose track on what version you are changing. Follow the steps set out below. You should use **Whole Word Replace** functionality for the following steps.

- 1 You should do a global find and replace in your Witango files of 'METAHTML' and replace with 'META'.
- 2 You should do a global find and replace in your Witango files of MULTILINEHTML and replace with W5_MULTILINEHTML.
- 3 You should do a global find and replace in your Witango files of MULTILINE and replace with W5_MULTILINE.
- 4 You should do a global find and replace in your Witango files of W5_MULTILINEHTML and replace with MULTILINE.
- 5 You should do a global find and replace in your Witango files of W5_MULTILINE and replace with MULTILINEHTML.

Licensing File Changes

The Witango Server 5.5 License information is made up of two parts:

- a license key; and
- a license file.

You will receive both of these via email when you activate your PIN on the witango.com web site.

Entering a new License File

To enter your license key follow the steps below:

- 1 Open the `witango.ini` file in a text editor. It is located in the following directory:

```
$WITANGO_PATH/configuration/witango.ini
```

- 2 Search the line `LICENSE=` and paste your serial number immediately after `"="`.



Note If you can not find such a line in the file, simply add a new line `LICENSE=your-serial_number` into the file.

- 3 **Save** the changes made to your `witango.ini` file.

To install the Witango License File copy the `license.ini` file to the configuration directory for your application:

```
$WITANGO_PATH/configuration/license.ini
```

Changing a License File

You may at some time need to change your Witango Server 5.5 license information, for example, going from Trial to Standard mode. To do this:

- 1 Open the `witango.ini` file in a text editor. It is located in the following directory:

```
$WITANGO_PATH/configuration/witango.ini
```

- 2 Search the line `LICENSE=` and paste your new serial number immediately after `"="`. Then **Save** your changes.

- 3 Copy your new `license.ini` file over the old file in the configuration directory for your application:

```
$WITANGO_PATH/configuration/license.ini
```

