TeraScript Programmer's Reference

TeraScript 6.2.3

January 2013

Tronics Software LLC
503 Mountain Ave.
Gillette, NJ 07933
USA

Email: support@terascript.com
Web: http://www.terascript.com

Table of Contents

1 **Table of Contents i**

2 **Introduction 1**

Conventions used in this manual 2

3 **Meta Tags 3**

Where You Can Use Meta Tags 4

Format of Meta Tags 5

- Syntax 5
- Naming Attributes 6
- Quoting Attributes 6

Encoding Attribute 8

- NONE 8
- HTML 8
- META 8
- METAHTML 8
- MULTILINE 9
- MULTILINEHTML 9
- URL 9
- JAVASCRIPT 9
- SQL 9
- CDATA 10

Format Attribute 11

Array-to-Text Attributes 17

<@ABSROW> 18

<@ACTIONRESULT> 19

<@ADDROWS> 20

<@APPFILE> 22

<@APPFILENAME> 23

<@APPFILEPATH> 24

<@APPKEY> 25

<@APPNAME> 26

<@APPPATH> 27

<@ARG> 28

<@ARGNAMES> 29

<@ARRAY> 30
Examples 31
<@ASCII> 32
<@ASSIGN> 33
<@BIND> 39
<@BREAK> 42
<@CALC> 43
<@CALLMETHOD> 55
<@CGI> 58
<@CGIPARAM> 59
<@CHAR> 60
<@CHOICELIST> 61
<@CIPHER> 66
<@CLASSFILE> 71
<@CLASSFILEPATH> 72
<@CLEARERRORS> 73
<@COL> 74
<@COLS> </@COLS> 75
<@COLUMN> 76
<@COMMENT> </@COMMENT> 77
<@CONFIGPATH> 78
<@CONNECTIONS> 79
<@CONTINUE> 82
<@CREATEOBJECT> 83
<@CRLF> 85
<@CURCOL> 86
<@CURRENTACTION> 87
<@CURRENTDATE>, <@CURRENTTIME>, <@CURRENTTIMESTAMP> 88
<@CURROW> 89
<@CUSTOMTAGS> 90
<@DATASOURCESTATUS> 91
<@DATEDIFF> 94
<@DATETOSECS>, <@SECSTODATE> 95

<@DAYS> 97
<@DBMS> 98
<@DEBUG> </@DEBUG> 99
<@DEFINE> 100
<@DELROWS> 103
<@DISTINCT> 105
<@DOCS> 108
<@DOM> 109
109
<@DOMAIN> 110
<@DOMDELETE> 111
<@DOMINSERT> 114
<@DOMREPLACE> 116
<@DOMSEARCH> 118
<@DQ>, <@SQ> 120
<@DSDATE>, <@DSTIME>, <@DSTIMESTAMP> 121
<@DSNUM> 123
<@DSTYPE> 124
<@EDITION> 125
<@ELEMENTATTRIBUTE> 126
<@ELEMENTATTRIBUTES> 129
<@ELEMENTNAME> 131
<@ELEMENTVALUE> 134
<@EMAIL> 137
<@EMAILSESSION> 140
<@ERROR> 143
<@ERRORS> </@ERRORS> 145
<@EXCLUDE> </@EXCLUDE> 146
<@EXIT> 147
<@FILTER> 148
<@FOR> </@FOR> 151
<@FORMAT> 153
<@GETPARAM> 154
<@HTTPATTRIBUTE> 156

<@HTTPREASONPHRASE> 159
<@HTTPSTATUSCODE> 160
<@IF>, <@ELSEIF>, <@ELSEIFEMPTY>, <@ELSEIFNO-TEMPTY>,<@ELSEIFEQUAL>, </@IF> 161
<@IFEMPTY> <@ELSE> </@IF> 165
<@IFEQUAL> <@ELSE> </@IF> 166
<@IFNOTEMPTY> <@ELSE> </@IF> 168
<@INCLUDE> 170
<@INTERSECT> 171
<@ISALPHA> 174
<@ISALPHANUM> 175
<@ISDATE>, <@ISTIME>, <@ISTIMESTAMP> 176
<@ISDECIMAL> 180
<@ISINT> 181
<@ISMETASTACKTRACE> 182
<@ISNULLOBJECT> 183
<@ISNUM> 184
<@KEEP> 185
<@LDAPADD> 186
<@LDAPDELETE> 188
<@LDAPMODIFY> 190
<@LDAPSEARCH> 192
<@LEFT> 195
<@LENGTH> 196
<@LITERAL> 197
<@LOCATE> 198
<@LOGMESSAGE> 199
<@LOWER> 200
<@LTRIM> 201
<@MAKEPATH> 202
<@MAP> 203
<@MAXROWS> 206
<@METAOBJECTHANDLERS> 207
<@METASTACKTRACE> 208

<@MIMEBOUNDARY> 209
<@NEXTVAL> 210
<@NULLOBJECT> 211
<@NUMAFFECTED> 212
<@NUMCOLS> 213
<@NUMOBJECTS> 214
<@NUMROWS> 215
<@OBJECTAT> 216
<@OBJECTS></@OBJECTS> 217
<@OMIT> 219
<@PAD> 220
<@PLATFORM> 221
<@POSTARG> 222
<@POSTARGNAMES> 223
<@PRODUCT> 224
<@PURGE> 225
<@PURGECACHE> 226
<@PURGEDEBUG> 228
<@PURGERESULTS> 229
<@RANDOM> 230
<@REGEX> 231
<@RELOADCONFIG> 233
<@RELOADCUSTOMTAGS> 234
<@REPLACE> 235
<@RESULTS> 237
<@RIGHT> 238
<@ROWS> </@ROWS> 239
<@RTRIM> 241
<@SCRIPT> 242
<@SEARCHARG> 245
<@SEARCHARGNAMES> 246
<@SECSTODATE>, <@SECSTOTIME>, <@SECSTOTS> 247
<@SERVERNAME> 248

<@SERVERSTATUS> 249
<@SETCOOKIES> 252
<@SETPARAM> 253
<@SLEEP> 255
<@SORT> 256
<@SQ> 258
<@SQL> 259
<@STARTROW> 260
<@STOP> 261
<@SUBSTRING> 262
<@THROWERROR> 263
<@TIMER> 265
<@TIMETOSECS>, <@SECSTOTIME> 266
<@TMPFILENAME> 267
<@TOGMT> 268
<@TOKENIZE> 269
<@TOTALROWS> 271
<@TOUTC> 272
<@TRANSPOSE> 273
<@TRIM> 274
<@TSTOSECS>, <@SECSTOTS> 275
<@UNION> 277
<@UPPER> 280
<@URL> 281
<@URLDECODE> 286
<@URLENCODE> 287
<@USERREFERENCE> 288
<@USERREFERENCEARGUMENT> 289
<@USERREFERENCECOOKIE> 290
<@UUID> 291
<@VAR> 292
Shorthand for XPATH and XPOINTER 295
<@VARINFO> 298
<@VARNAMES> 300

<@VARPARAM> 301
<@VERSION> 302
<@WEBROOT> 303
<@WHILE> </@WHILE> 304
<@XSLT> 306
<@!> 309

4 Custom Meta Tags 311

Using Custom Meta Tags 312
- Attributes of Custom Meta Tags 312
- Tag Name Conflicts 312
- Custom Meta Tag Limitations 312

Creating Custom Meta Tags: Tag Definition File 313
- Custom Tag Definition File Format 313

Loading Tags 316
- Reloading Custom Meta Tags 316
- Retrieving Information on Custom Meta Tags 316

Installing Custom Meta Tag Definition Files 317
- Application-specific Custom Meta Tags 317

Custom Meta Tag Example: tabletag.xml 318

Custom Tag Generator 323

5 Working With Variables 325

About Variables 326
- Naming Variables 326
- Variable Types 326
- Understanding Scope 327
- Basic TeraScript Scopes 328
- TeraScript Class File-only Scopes 333
- Custom Scopes 334
- Returning Variable Values 335
- Purging Variables 336
- Arrays 336
- How TeraScript Determines Default Scope in Variable Assignments 340

Using Configuration Variables 342

Using User Keys 344

Changing the User Key 347

Assigning Values to userKey and altUserKey 347
Alternate User Keys 347
Returning the Value of userKey and altUserKey 348
Using Application File User Keys 348

6 Document Object Model 349

What is DOM? 350

Overview of Using DOM 352

Example 353

XPointer Syntax 355

Root 355
ID 355
Child 355
Descendant 355
Terms of Child or Descendant 356
Example 357

XPath Syntax 359

XPath Functions 359
Example 360

Manipulating a Document Instance 362

Creating a Document Instance 362
Using DOM Meta Tags 363

Returning XML in TeraScript Applications 365

Using <@VAR> and <@ASSIGN> With DOM 365
Using <@ELEMENT...> Meta Tags With DOM 368

Applications of DOM 371

Creating Complex Data Structures 371
Separating Business and Presentation Logic 373
Reading and Writing TeraScript Application Files 375
Other Uses 376

7 Configuration Variables 379

A Note on Scope 380

A Note on Default Locations 381

Alphabetical List of Configuration Variables, With Scopes 382

absolutePathPrefix 386

altUserKey 387

appConfigFile 388

applicationSwitch 389
aPrefix 390
aSuffix 391
cache 392
cacheIncludeFiles 393
cacheSize 394
cDelim 395
configPasswd 396
cPrefix 397
crontabFile 398
Format of the crontab File 398
Example of the crontab File 399
cSuffix 401
currencyChar 402
customScopeSwitch 404
customTagsPath 405
dataSourceLife 406
dateFormat, timeFormat, timestampFormat 407
DBDecimalChar 410
debugMode 411
decimalChar 412
defaultErrorFile 414
defaultScope 415
docsSwitch 416
domainConfigFile 417
domainScopeKey 418
DSConfig 419
DSConfigFile 421
encodeHTTPResponse 423
encodeResults 424
encodeResultsHTML 425
externalSwitch 426
fileDeleteSwitch 427
fileReadSwitch 428

fileWriteSwitch 429
headerFile 430
httpHeader 431
javaScriptSwitch 432
javaSwitch 433
license 434
licenseErrorHTML 435
listenerPort 436
lockConfig 437
logArguments 438
logDir 439
loggingLevel 440
logToResults 441
mailAdmin 442
mailDefaultFrom 443
mailPort 444
mailServer 445
mailSwitch 446
maxActions 447
maxSessions 448
noSQLEncoding 449
objectConfigFile 450
ociLibPath 451
odbcDmlLibrary 452
passThroughSwitch 453
persistentRestart 454
pidFile 455
postArgFilter 456
queryTimeout 457
rDelim 458
requestQueueLimit 459
returnDepth 460
rPrefix 461
rSuffix 462

sendUserReferenceCookie 463
shutdownUrl 464
startStopTimeout 465
startupUrl 466
staticNumericChars 467
stripCHARs 468
TCFSearchPath 469
thousandsChar 470
threadPoolSize 472
timeFormat 473
timeoutHTML 474
timestampFormat 475
useFullPathForIncludes 476
userAgent 477
userKey, altuserKey 478
validHosts 480
varCachePath 481
variableTimeout 482
variableTimeoutTrigger 483
webServices 484
webServicesExtns 485

8 TeraScript Server Error Codes 487

9 Web Services 495

Web Services Overview 496
Enabling Web Services 499
Register a file extension 500
Creating a WSDL file for your tcf 501
Configuring the TeraScript Server for Web Services 506
- Sample web service configuration 506
- SOAP related variables and parameters 506

How to consume your TeraScript Web Service 507
- Creating a Simple Web Service 507

Calling the Web Service via HTTP 511
Calling the web service via a TeraScript File 513

10 <@CALC> Expression Operators 515

Numbers 516
Hexadecimal, Octal and Binary Numbers 518
Strings 519
Calculation Variables 521
Operators 522
Built-in Functions 524
Array Operators 526

11 Lists of Meta Tags 529

Alphabetical List of Meta Tags 530
Alphabetical List of Meta Tags, With Attributes 538
Meta Tags List by Function 543

12 Using DLLs With TeraScript 549

TExtParamBlock 550
DLL Functions 551

CHAPTER 1

Introduction

1

An Overview of This Manual

This manual provides detailed explanations of TeraScript Meta Tags, which are used to construct TeraScript application files and TeraScript class files, and also provides details on configuration variables, which are used to configure TeraScript Server.

- Meta Tags: Chapter 2 (page 3), including Custom Meta Tags (page 305).
- Configuration Variables: Chapter 7 (page 373).

This manual is intended as a reference for users who are familiar with TeraScript.

Some topics in this manual may apply only to Mac OS X, Windows, or Linux.

The Mac™ OS X, Microsoft™ Windows™, and Linux™ graphics identify those topics, respectively; otherwise, topics apply equally to all platforms.

Conventions used in this manual

TERASCRIPT_PATH is used throughout this document to indicate the filepath to where the TeraScript Server executable is located on the machine, eg:

For Windows (32-bit):

```
C:\Program Files\Tronics Software\TeraScript Server 6
```

For Windows (64-bit):

```
C:\Program Files (x86)\Tronics Software\TeraScript Server 6
```

For Linux:

```
/usr/local/TeraScript Server 6
```

For Mac OS X:

```
/Applications/TeraScript Server 6
```

CHAPTER 2

Meta Tags

2

TeraScript Meta Tags Reference

Meta Tags are the components of a markup language that is interpreted by TeraScript Server. This language is similar in form to HTML but much more dynamic.

Meta Tags are resolved by TeraScript Server when your application file is executed.

This chapter covers the following topics:

- where you can use Meta Tags
- basic Meta Tag syntax
- the ENCODING attribute
- the FORMAT attribute
- array-to-text attributes
- a detailed look at each Meta Tag.

Where You Can Use Meta Tags

Most Meta Tags can be used in all places in application files where text or HTML can be inserted, including these application file locations:

- attribute HTML that is attached to an action, including:
  - Results HTML
  - Error HTML
  - No Results HTML
- actions in an application file, including:
  - parameters in Search, Update, and Delete actions
  - column values in Update and Insert actions
  - custom column references used in database actions
  - Maximum Matches and Start Match fields in Search and Direct DBMS actions
  - External action parameters
  - Assign actions (both name and value)
  - If/Else If action parameters
  - SQL entered into the Direct DBMS action window
  - parameters in Mail, File System, and Web Call actions
- files included using the `<@INCLUDE>` Meta Tag
- most attributes for other Meta Tags.

Where you can insert Meta Tags, the contextual menu (accessible from a right mouse click on Windows or a CONTROL click on Macintosh) shows **Insert Meta Tag**.

Format of Meta Tags

Syntax

The basic syntax for TeraScript Meta Tags is:

```
<@METATAG ATTRIBUTENAME="ATTRIBUTEVALUE">
```

- The opening "<" is a characteristic of tag languages, including HTML. The "@" symbol distinguishes TeraScript Meta Tags. No spaces are allowed between the opening "<", the "@", and the first character of the Meta Tag name.

  At least one space must occur between the Meta Tag name and the first attribute name, and between all attribute values and subsequent attribute names. For example:

  ```
  <@POSTARG NAME="Bruce" ENCODING="NONE">
  ```

  and

  ```
  <@POSTARG       NAME="Bruce"   ENCODING="NONE">
  ```

  are both valid Meta Tag syntax.

- Line breaks are allowed in Meta Tags anywhere a space occurs. For example:

  ```
  <@ASSIGN
  NAME="varname"
  SCOPE="request"
  VALUE="somevalue"
  >
  ```

  is valid TeraScript Meta Tag syntax.

- There is no space allowed before or after the equals (=) sign between an attribute name and its value. As well, if you quote the value of an attribute, no space is allowed between the equals (=) sign and the opening quote. For example,

  ```
  <@POSTARG NAME="Bruce">
  ```

  is correct syntax.

- This documentation shows Meta Tags in uppercase, but Meta Tags are case insensitive. That is, all of the following are valid TeraScript Meta Tag syntax:

  ```
  <@CALC EXPR="3+7">
  <@Calc expr="3+7">
  <@calc Expr="3+7">
  ```

Naming Attributes

TeraScript uses named attributes in most Meta Tags. All attributes have a name. The order of the attributes used in a Meta Tag does not matter if the attributes are referenced by name; for example,

```
<@POSTARG NAME="foo" ENCODING="METAHTML">,
```

and

```
<@POSTARG ENCODING="METAHTML" NAME="foo">
```

are equivalent.

The name for every attribute you specify must be provided, with one exception: any attribute that is required—that is, any attribute whose absence makes a Meta Tag invalid—can be specified without a name, as long as it occurs in its predefined position (usually immediately following the name of the Meta Tag).

Note The documentation in this chapter shows Meta Tag syntax with the required order for positional (required) attributes.

`<@POSTARG homer>` is valid in TeraScript, because the `NAME` attribute is required, and its designated position is first (immediately following the Meta Tag). If you want to specify the encoding, you must use `<@POSTARG homer ENCODING="NONE">`, because `ENCODING` is not a required attribute. For new users of TeraScript, the best method to adopt is to enter all attribute names, for example, `<@POSTARG NAME="homer" ENCODING="NONE">`.

Quoting Attributes

Attribute values must sometimes be quoted to avoid ambiguity. For example, whenever you need to specify an attribute value that includes a space, you must put quotes around it. To refer to a database column called "Zip Code", for example, use `<@COLUMN NAME="Zip Code">`. Without the quotes, `<@COLUMN NAME=Zip Code>`, TeraScript would incorrectly interpret `Zip` as the attribute name and `Code` as the start of the next attribute. TeraScript recognizes both the double (") and single (') quote character pairs as attribute delimiters.

Another case where quotes are necessary is when specifying an empty value for the attribute ("" tells TeraScript that there is no

value). <@ASSIGN NAME=myVar VALUE=""> will assign an empty string to the variable named myVar.

Note Quotes are not necessary when you are using only a Meta Tag as the attribute value. TeraScript knows that Meta Tags begin with <@ and end with >, so no quotes are necessary to delimit the value.

It is necessary, however, to quote attribute values if they contain any whitespaces, even if between Meta Tags, such as VALUE="<@CURRENTDATE> <@CURRENTTIME>"

In general, quoting attribute values is recommended. It is never incorrect to quote an attribute value.

Some additional rules to follow when quoting Meta Tag attributes are as follows:

For more information, see "<@DQ>, <@SQ>" on page 120.

- If you have a literal double or single quote in a Meta Tag attribute value, you must replace it with the <@SQ> or <@DQ> Meta Tag, regardless of which quote character is delimiting the attribute value.

For more information, see "<@CALC>" on page 43 and "<@IF>" on page 161.

- The exceptions to the last rule are the expressions specified for <@CALC>, <@IF>, and <@WHILE> Meta Tags, and the Advanced mode for If, Else If, and While Loop actions. The EXPR attribute can use quotes as part of an expression, as long as they are not the same quotes as surround EXPR. These quotes are taken as delimiters for individual values within the expression. The expression attribute also supports backslash-escaping of quotes: \" and \' for literal quotes (and require the use of \\ for \ as a result).

Alternating Quote Rule

If you have a nested tag in an attribute, use the "other" quote character around its value. This alternating can go on indefinitely for deeply nested tags. This allows you to distinguish between quotes you want to specify as part of the attribute value itself, instead of as an attribute delimiter. For example:

```
<@ARG NAME="<@VAR NAME='<@VAR NAME="myArgNameVar">'>">
```

Encoding Attribute

Many value-returning Meta Tags accept an optional ENCODING attribute that determines how returned values are formatted. Each of the valid format types is described in this section.

The default ENCODING method for all actions and Meta Tags executed by the TeraScript Server is **NONE** . This means that the text returned by the Meta Tag is not encoded before it is sent to the user's web browser.

NONE

The NONE value for the ENCODING attribute allows you to indicate that the value returned by the Meta Tag contains HTML formatting codes that are to be passed back to the user's Web browser and processed as HTML by the browser. This is the typical use for most Meta Tags, and is the default behavior.

For example:

```
<@COLUMN NAME="pages.theHTMLpage" ENCODING="NONE">
```

HTML

The HTML value for the ENCODING attribute allows you to indicate that the value returned by the Meta Tag contains HTML formatting codes that are to be translated as they are passed back to the user's Web browser therefore displaying the HTML codes to the user. For example, < is changed to <.=

META

The META attribute value of the ENCODING attribute performs the same function as NONE but also looks for TeraScript Meta Tags in the value and evaluates any it finds.

METAHTML

The METAHTML attribute value of the ENCODING attribute performs the same function as HTML but also looks for TeraScript Meta Tags in the value and evaluates any it finds.

For example:

```
<@COLUMN NAME="table.template" ENCODING="METAHTML">
```

If the template column contains the text <@VAR NAME="foo">, the example shown returns the current value of the variable foo.

MULTILINE The MULTILINE attribute value causes TeraScript to replace return, line feed, and return/line feed combinations in the value with <br /> tags. It will not do HTML encoding.

MULTILINEHTML The MULTILINEHTML attribute value lets you combine the functions of HTML and MULTILINE.

URL The URL formatting attribute value tells TeraScript to make the value returned by a Meta Tag safe for inclusion in a URL by encoding special characters such as spaces and slashes, according to the scheme set out in RFC 1630. The main use for this attribute value is to construct URLs containing database or user-entered values.

For example:

```
<A HREF="/customer_detail?cust_name=
<@COLUMN NAME='customer.cust_name'
ENCODING='URL'>">
More customer info</A>
```

If the URL attribute were not used in this case, links to customer names from the database that contained spaces would not work properly because a space is invalid in a URL. By using the URL attribute value, any spaces are converted to %20. Similarly, other special characters that have meaning in URLs (~, #, and so on) are also converted.

The <@URLENCODE> Meta Tag performs the same function on any value. It is strongly recommended that you get into the habit of encoding any Meta Tags included in a URL, even if you think the value returned is not going to require it.

JAVASCRIPT Encodes the value to make it a valid JavaScript literal. It does this by escaping certain characters using a backslash; for example, tabs are converted to \t. Use this type of encoding when using a Meta Tag in server- or client-side JavaScript code.

SQL The SQL encoding type converts the specified value by doubling all occurrences of the single quote character.

TeraScript Server automatically performs SQL encoding on Meta Tag values substituted in Direct DBMS SQL, except when the configuration variable noSQLEncoding is set to true. The SQL ENCODING attribute value is generally appropriate only when noSQLEncoding is set to true, and allows you to toggle SQL encoding on or off for particular Meta Tags.

For example:

```
<@ASSIGN NAME='mySQL' VALUE='SELECT * FROM customer
WHERE cust_name=<@SQ><@ARG "cust_name"
ENCODING="sql"><@SQ>'>
```

CDATA

CDATA (Character Data) is a keyword used in SGML and XML to indicate blocks of text that are not to be parsed, even if they contain markup. This contrasts with PCDATA (Parsed Character Data).

Values encoded as CDATA may contain any valid character data; tags may be included in the value, but they are not to be recognized by the XML or SGML parser, and are not processed as tags normally are.

In TeraScript, ENCODING=CDATA is most often used in conjunction with the DOM Meta Tags that parse XML (<@DOM> and <@DOMINSERT>) to ensure correct parsing of data.

In general, encoding a value as CDATA simply results in <![CDATA[valuegoeshere]]> being returned. If the value contains the CDATA end sequence "]]>", the text is broken up into CDATA/PCDATA/CDATA for each occurrence, to ensure proper parsing. This special processing must be done, or the CDATA end sequence in the value would cause the CDATA block to end prematurely. For example, if you have a variable, fred, containing <[[test]]>, the following results in a parsing error:

```
<@DOMINSERT OBJECT="foo">
<TEST><![CDATA[<@VAR fred>]]></TEST>
<@DOMINSERT>
```

You can parse this data properly with the CDATA encoding type:

```
<@DOMINSERT OBJECT="foo">
<TEST><@VAR fred ENCODING="CDATA"></TEST>
<@DOMINSERT>
```

Note It is because your data might contain the]]> sequence that you should use the CDATA encoding type, rather than simply using <![CDATA [my data]]>.

Format Attribute

The FORMAT attribute is optional with many Meta Tags. It specifies how the output of the tag should be formatted.

All tags with an optional FORMAT attribute accept a format string of the form `FORMAT=class:format`, as detailed following.

CASE: Case Reformatting

Text can be converted as follows:

- to uppercase with **case:upper** (for example, `HEllo` - `HELLO`)
- to lowercase with **case:lower** (for example, `HEllo` - `hello`)
- to wordcase with **case:word** (for example, `HEllo` - `Hello`).

Words are defined as a sequence of non-whitespace characters delimited by whitespace.

NUM: Numeric Formatting

Numbers with at least one whole digit and optional fractional digits can be reformatted.

The format is specified by an ordered, comma-delimited list of values, as such:

```
FORMAT=num:1,2,3,4,5,6,7,8
```

The following table describes the format of the values that must go in each position.

#	Value
1	Grouping Defines how you want your numbers grouped, for example, in groups of three. There are two ways to apply your grouping. They are: A hyphen-delimited list of digits from the least-significant place (beginning from the right). An asterisk (*) means repeat using the last specification; no number means the output remainder is untouched. Examples: 3-*1,234,567,890,123 This example repeats a grouping of "3". 3-1-21234567,89,0,123 This example groups six digits, beginning from the right, into three separate groups, one of three ("3") digits, one of one ("1"), and one of two ("2").
2	Grouping separator Defines the character that separates the groupings. In the previous example, it is a comma. It may be multiple characters.
3	Fractional Digits Defines the number of fractional digits to show: If a number is specified, that many digits are displayed; the value is truncated or 0-padded as appropriate. If no number is specified, then the number of fractional digits passed in is displayed untouched.
4	Fraction Separator Defines the fraction separator which may be multiple digits, and it is displayed even if no fractional digits are present. This is similar to a decimal place separator.
5	Positive Prefix Determines the prefix used if the number is positive (for example, +).
6	Positive Suffix Determines the suffix used if the number is positive.
7	Negative Prefix Determines the prefix used if the number is negative (for example, -).
8	Negative Suffix Determines the suffix used if the number is negative.

All eight items must be present, either with a specified value or nothing. For example, if you do not want any formatting for fractional digits there would be nothing between the commas, (,,).

You must address each of the eight items in the number formatting string, even if it is just to let TeraScript know not to do anything with one or more of the eight numerical formatting items in the list.

Each list item may be a maximum of 15 characters long. Spaces and case are significant.

Do not include spaces unless they are intended. Commas, single quotes, and double quotes can be included by backslash-escaping (for example, \, or \"), or enclosing them in single or double quotes (for example, "," or '"). Anything following a backslash is taken literally.

Here are some examples of numeric formats. -**1234.56** is formatted as:

Format	Format String	Example
Swiss Monetary	num:3-*,\,,2,.,SFrs.,,SFrs.,C	SFrs.1,234.55C
US Accounting	num:3-*,\,,2,.,$,,($,)	$(1,234.56)
French language numerals	num:3-*, ,3,\,,,,'- ',	- 1 234,560
credit sheet	num:,,,.,Balance: , credit,Balance: , debit	Balance: 1234.56 debit

Synonyms

There are synonyms provided for commonly used format strings.

Example synonyms are listed in the table following with formatted string, **-1234567.890**.

Format *	Equivalent Format	Sample Output
num:CA-accounting num:US-accounting	num:3-*,\,,2,.,$,,($,),	($ 1,234,567.89)
num:comma-float	num:3-*,\,,,.,,,,-,	-1,234,567.890
num:comma-integer	num:3-*,\,,0,,,,-,	-1,234,567
num:simple-float	num:,,,.,,,,-,	-1234567.890
num:simple-integer	num:,,0,,,,-,	-1234567

*US = United States and CA=Canada.

TEL: Telephone Numbers

This formatting accepts as input text a sequence of digits, spaces, or punctuation marks, and outputs the digits in one of the following requested formats:

Format*	Sample Output	Input Restrictions
tel:US-short tel:CA-short	819-1173	7-11 digits required
tel:US-long tel:CA-long	(905) 819-1173	10-11 digits required
tel:US-intl tel:CA-intl	+1 905 819-1173	10-11 digits required
tel:US-hyphen tel:CA-hyphen	1-905-819-1173	10-11 digits required

*US = United States and CA = Canada

DATETIME

The `format` attribute accepts the "`%-`" specifiers used by the `dateFormat` configuration variables, with the addition of a `datetime:` prefix. For example, `datetime:%Y-%m-%d` would specify an ODBC-style date (December 1st, 2010 would be formatted as "2010-12-01").

For more information, see "dateFormat, timeFormat, timestampFormat" on page 401.

TeraScript attempts to guess what the date/time entered actually is. First, the `dateFormat`, `timeFormat`, and `timestampFormat` configuration variables are used to test the input string for a perfect match, and failing these, the procedures as used by the `<@ISDATE>` family of tags are tested. If the input cannot be determined, a warning is logged and reformatting does not take place.

Tags that accepted a format attribute in previous versions of TeraScript—`<@CURRENTTIME>`, `<@CURRENTIMESTAMP>`, and `<@CURRENTDATE>`—can be used with the new FORMAT attribute or with their old formatting.

There are six datetime-class synonyms, as per the chart below:

Synonym	Equivalent Format	Sample Output
datetime:http	datetime:%a, %d %b %Y %H:%M:%S UTC	Fri, 15 Jun 2012 00:42:13 UTC

Synonym	Equivalent Format	Sample Output
datetime:rfc850	datetime:%A, %d-%b-%y %H:%M:%S GMT	Friday, 15-Jun-12 00:42:13 GMT
datetime:email	datetime:%a, %d %b %Y %H:%M:%S +0000	Fri, 15 Jun 2012 00:42:13 +0000
datetime:sql-date	datetime:{ d '%Y-%m-%d' }	{ d '2012-05-11' }
datetime:sql-time	datetime:{ t '%H:%M:%S' }	{ t '00:42:13' }
datetime:sql-ts	datetime:{ ts '%Y-%m-%d %H:%M:%S' }	{ ts '2012-05-11 00:42:13' }

Note `datetime:http`, datetime:rfc850, and datetime:email formatting do not make any adjustments to the time value to correct to GMT time. They simply output the input timestamp in the specified format. To convert a local time to GMT, use <@TOGMT>. <@TOUTC> is a synonym of <@TOGMT>, and <@TOGMT> is a synonym for <@TOUTC>.

For more information, see "<@TOGMT>" on page 268.

Date and Time Formatting Codes

Code	Description
%a	abbreviated weekday name
%A	full weekday name
%b	abbreviated month name
%B	full month name
%c	local date and time representation
%d	day of month (01–31)
%H	hour (24 hour clock)
%I	hour (12 hour clock)
%j	day of the year (001–366)
%m	month (01–12)
%M	minute (00–59)
%p	local equivalent of AM or PM
%S	second (00–59)
%U	week number of the year (Sunday= first day of week) (00–53)
%w	weekday (0–6, Sunday is zero)
%W	week number of the year (Monday = first day of week) (00–53)
%x	local date representation
%X	local time representation
%y	year without century (00–99)
%Y	year with century
%%	% sign

Array-to-Text Attributes

An array returned by a Meta Tag is converted to text when it is being returned to a Web browser. However, array-returning Meta Tags return an array when an array is copied from one place to another; for example, if an array-returning Meta Tag is used within <@ASSIGN>, no conversion to text is performed.

Some Meta Tags, such as <@VAR>, also accept a TYPE attribute, which, when set equal to 'text', forces an array to be returned as text. In that case, assigning the value of the array to a variable assigns a text representation of the array, with all the array, row, and column prefixes and suffixes described below.

TeraScript Meta Tags that return arrays take a series of optional attributes that allow you to format the text representation of the array. There are corresponding configuration variables, with the same names, whose values are used for array formatting if these attributes are not specified. The attributes are given in the following table:

Attribute	Description
APREFIX	The array prefix string. Default value: <table>
ASUFFIX	The array suffix string. Default value: </table>
RPREFIX	The row prefix string. Default value: <tr>
RSUFFIX	The row suffix string. Default value: </tr>
RSEP	Value placed between each row. No default value.
CPREFIX	The column prefix string. Default value: <td>
CSUFFIX	The column suffix string. Default value: </td>
CSEP	Value placed between each column. No default value.

Far more information, see aPrefix <page $pagenum>, aSuffix <page $pagenum>, cPrefix <page $pagenum>, cSuffix <page $pagenum>, rPrefix <page $pagenum> and rSuffix <page $pagenum>.

These attributes are used for defining the appropriate text for display, before and after the specific components of the array are displayed. This is useful for automatically displaying the contents of arrays as tables (the default) or ordered lists.

Meta Tags that return arrays are specified in this manual with {*array-to-text attributes*} in the syntax specification of the Meta Tag. When using Meta Tags that return arrays, you can specify any or all of the array-to-text attributes to override the default values, shown above, when the array is returned as text.

<@ABSROW>

Description

Returns the position of the current row within the total rowset matched by a Search action's criteria.

When used outside of the <@ROWS></@ROWS> block of a Search or Direct DBMS action's Results HTML, this Meta Tag returns zero.

Example

```
<P>There are <@TOTALROWS> records matching your
criteria. Here are records <@STARTROW> through <@CALC
EXPR="<@STARTROW>+<@NUMROWS>-1">:</P>

<@ROWS>
<P>Here is matching record number <@ABSROW>:
<P><STRONG>Name:</STRONG>
<@COLUMN NAME="contact.name" FORMAT="case:upper">
<STRONG>Phone:</STRONG> <@COLUMN
NAME="contact.phone" FORMAT="tel:CA-short">
</@ROWS>
```

This HTML displays the match number for each record displayed, relative to the first matching record.

See Also

<@CURROW>	page 89
<@NUMROWS>	page 215
<@ROWS> </@ROWS>	page 239
<@STARTROW>	page 260
<@TOTALROWS>	page 271

<@ACTIONRESULT>

Syntax

<@ACTIONRESULT NAME=*actionName* NUM=*itemNumber* [ENCODING=*encoding*] [FORMAT=*format*]>

Description

This Meta Tag has been deprecated as of the release of TeraScript 6. It is currently operational but will be removed in the next major version release. Developers are encouraged to discontinue use of this Meta Tag. When applicable, a warning will be reported to the witangoevents.log file.

Returns the value of the specified item from the first row of results generated by an action in the current execution.

Use this Meta Tag inside any action to reference data from the first row of a previously executed results generating action, such as a Search, External, or Direct DBMS action. The NAME attribute refers to the name of the action that generated the result during the current execution of the application file. The NUM attribute is the number of the column to get (for example, to get the value of the third column in the first row returned by an action, specify NUM=3).

Note When the action result being asked for has been executed multiple times, as can occur if the action is inside a loop, the value from the last execution of the action is returned. When the action name specified is ambiguous, as can occur when branching to another application file, the <@ACTIONRESULT> tag refers to the last one executed.

Example

```
Your new account number is:
<B><@ACTIONRESULT NAME="GetUniqueID" NUM="1"></B>.
```

In this example, <@ACTIONRESULT> evaluates to the first item from the first row of the result set generated by the action GetUniqueID.

See Also

<@COL>	page 74
<@COLUMN>	page 76
Encoding Attribute	page 8
<@FORMAT>	page 153
Format Attribute	page 11
<@PURGERESULTS>	page 229
<@RESULTS>	page 237

<@ADDROWS>

Syntax

<@ADDROWS ARRAY=*arrayVarName* VALUE=*rowsToAdd* [POSITION=*position*] [SCOPE=*scope*]>

Description

Adds the rows specified in VALUE to the array in the variable named by ARRAY. This tag does not return anything.

If the variable specified by the ARRAY attribute does not exist, it is created.

The VALUE attribute specifies the row(s) to add. You may use the <@VAR> tag and specify a variable containing an array, or specify any other Meta Tag that returns an array. This array must have the same number of columns as the one specified by ARRAY; otherwise, an error is generated.

For single-column arrays, the VALUE attribute may be a text value, rather than an array. In this case, a single row is added with the value specified.

The POSITION attribute specifies the index of the row to start adding from; the rows are added after the specified row. To add rows to the beginning of the array, use 0 as the value for POSITION. To add rows to the end of the array, use -1. If POSITION is not specified, the rows are added to the end.

The SCOPE attribute specifies the scope of the variable specified as the value of the ARRAY attribute. If the scope is not specified, the default scoping rules are used.

Meta Tags are permitted in any of the attributes.

Examples

- If the request variable `colors` contains the following array:

orange
amber
burnt umber

and the request variable `colors2` contains the following array:

yellow

<@ADDROWS ARRAY="colors" SCOPE="request" VALUE="@@request$colors2"> results in colors containing:

orange
amber
burnt umber
yellow

- If the user variable choices_list contains the following array:

News	2
Sports	3
Movies	4

and the user variable new_choices contains the following array:

Stocks	1
Weather	5

<@ADDROWS ARRAY="choices_list" SCOPE="user" VALUE="<@VAR NAME='new_choices' SCOPE='user'>" POSITION=1> results in choices_list containing:

News	2
Stocks	1
Weather	5
Sports	3
Movies	4

See Also

<@DELROWS> page 103
<@UNION> page 277

<@APPFILE>

Syntax

```
<@APPFILE [ENCODING=encoding]>
```

Description

Returns the path and name to the current application file. This Meta Tag is useful for creating links that reference the current application file. The path returned is always relative to the Web server root directory.

Note This Meta Tag is often used to create URLs, for example, in the HREF attribute of an anchor tag in HTML. To make sure that the Meta Tag returns a properly encoded value, you can use the following:

```
<@APPFILE ENCODING="URL">
```

Example

```
<A HREF="<@CGI><@APPFILE>?conf=<@COLUMN NAME=
'conferences.conf_id'>&function=messages">
<@COLUMN NAME="conferences.conf_name"> Messages </A>
```

This example specifies a link to the current application file.

See Also

<@APPFILEPATH>	page 24
<@CGI>	page 58
Encoding Attribute	page 8
<@MAKEPATH>	page 202

<@APPFILENAME>

Syntax

```
<@APPFILENAME [ENCODING=encoding]>
```

Description

Returns the name of the current application file. This Meta Tag is useful for creating links that reference the current application file.

Compare the following two tags:

- <@APPFILE> returns the path and the name
- <@APPFILEPATH> returns the path and not the name.

Note This Meta Tag is often used to create URLs, for example, in the HREF attribute of an anchor tag in HTML. To make sure that the Meta Tag returns a properly encoded value, you can use one of the following:

```
<@APPFILE ENCODING="URL">
```

Example

The following example processes different HTML depending on the name of the current application file. You may find this useful in files that are referenced with <@INCLUDE> that are used by several application files but that you would like to behave differently in different application files.

```
<@IFEQUAL <@APPFILENAME> customers.taf>
 [...HTML to execute...]
<@ELSEIFEQUAL <@APPFILENAME> administrator.taf>
 [...HTML to execute...]
<@ELSE>
 [...default HTML to execute...]
<@/IF>
```

See Also

<@APPFILE>	page 22
<@APPFILEPATH>	page 24
Encoding Attribute	page 8
<@INCLUDE>	page 170
<@MAKEPATH>	page 202

<@APPFILEPATH>

Syntax

```
<@APPFILEPATH [ENCODING=encoding]>
```

Description

Returns the path to the current application file, excluding the application file name, but including the trailing slash.

This Meta Tag is useful for creating links that reference an application file, <@INCLUDE> file, or image file in the same directory as the currently executing application file.

The path returned is always relative to the Web server root directory.

Note This Meta Tag is often used to create URLs, for example, in the HREF attribute of an anchor tag in HTML. To make sure that the Meta Tag returns a properly encoded value, you can use one of the following:

```
<@APPFILE ENCODING="URL">
```

Examples

```
<A
HREF="<@CGI><@APPFILEPATH>homer.taf?function=form">
</A>
```

This example calls the `homer.taf` file, located in the same directory as the currently executing application file.

```
<@INCLUDE FILE="<@APPFILEPATH>header.html">
```

This example includes the `header.html` file, located in the same directory as the currently executing application file.

```
<IMG SRC="<@APPFILEPATH>logo.gif">
```

This example references the `logo.gif` file, located in the same directory as the currently executing application file.

See Also

<@APPFILE>	page 22
<@CGI>	page 58
Encoding Attribute	page 8
<@INCLUDE>	page 170
<@MAKEPATH>	page 202

<@APPKEY>

Syntax

```
<@APPKEY [ENCODING=encoding]>
```

Description

This Meta Tag returns the key value of the current application scope. The tag returns nothing if the currently-executing TeraScript application file is not part of an application. Managing an application is done through the Administration Application `config.taf` or by editing the `applications.ini` file.

Example

The default value of the `userKey` configuration variable, which sets the key value for user scope, is:

```
<@APPKEY><@USERREFERENCE><@HTTPATTRIBUTE CLIENT_IP>
```

The presence of <@APPKEY> in the key means that the same variable name can be used in different applications without conflicting.

See Also

`<@APPNAME>`	page 26
`<@APPPATH>`	page 27
`applicationSwitch`	page 383
`appConfigFile`	page 382
`Encoding Attribute`	page 8
`userKey, altuserKey`	page 474

<@APPNAME>

Syntax

```
<@APPNAME [ENCODING=encoding]>
```

Description

This Meta Tag returns the name of the current application. This tag returns nothing if the currently-executing TeraScript application file is not part of an application. Managing an application, including setting its name, is done through the Administration Application `config.taf`.

Example

```
My Project
```

In this example, the Meta Tag is displaying "My Project" as the name of the current application.

See Also

`<@APPKEY>`	page 25
`<@APPPATH>`	page 27
`applicationSwitch`	page 383
`appConfigFile`	page 382
`Encoding Attribute`	page 8

<@APPPATH>

Syntax

```
<@APPPATH [ENCODING=encoding]>
```

Description

This Meta Tag returns the path to the current application, or nothing if the currently-executing application file is not part of an application. Managing an application, including setting its path, is done through the Administration Application `config.taf` .

Example

```
\TeraScript\my_project
```

In this example, the “my_project” application is stored in the “TeraScript” folder.

See Also

`<@APPKEY>`	page 25
`<@APPNAME>`	page 26
`applicationSwitch`	page 383
`appConfigFile`	page 382
`Encoding Attribute`	page 8

<@ARG>

Syntax

```
<@ARG NAME=name [TYPE=type] [FORMAT=format]
[ENCODING=encoding]>
```

Description

Returns the value(s) of the named search or post argument in the HTTP request that calls the application file. References to arguments not present in the request evaluate to empty.

Use this Meta Tag (rather than <@SEARCHARG> or <@POSTARG>) when you want the flexibility of passing a value to an application file via either a search or post argument.

The NAME attribute may be specified as a literal value, value-returning Meta Tag, or a combination of both. The TYPE attribute accepts one of two possible values: TEXT or ARRAY. ARRAY causes the tag to return a single-column, multi-row array of values, one for each value received for the named argument. An HTML <SELECT> form field with the MULTIPLE attribute, for example, sends multiple instances of the form field, one for each value selected by the user. Using the ARRAY type lets you access all those values.

TEXT, which is the default type if the TYPE attribute is not specified, causes the tag to return a single value. If you specify this type when multiple values were received for the argument, the value returned is the first one received by TeraScript.

The optional FORMAT attribute determines how the value is formatted by TeraScript; it is ignored if TYPE=ARRAY is specified.

Examples

```
<@ARG NAME="foo">
```

These return the value of the "foo" argument. Even if more than one value was specified for foo, only one is returned.

```
<@ARG NAME="foo" TYPE="ARRAY">
```

Returns an array containing all values for the "foo" argument.

See Also

Encoding Attribute page 8
Format Attribute page 11
<@POSTARG> page 222
<@SEARCHARG> page 245

<@ARGNAMES>

Syntax

<@ARGNAMES [{*array attributes*}]>

Description

Returns an array with two columns specifying all search and post arguments passed into the current application file. The first column contains the name of the argument, and the second column contains either POST or SEARCH, depending upon how the argument was sent to the server.

There are array-returning attributes that can be used to specify prefixes and suffixes for the returned array, rows within the array, and columns within the rows. They are described in the section Array-to-Text Attributes <page $pagenum>. By default, when used in at text context, the returned array is formatted as an HTML table.

Example

```
View the arguments <@ARGNAMES>.
```

This would return something like:

Fred	POST
access	POST
username	SEARCH

See Also

<@POSTARGNAMES> page 223
<@SEARCHARGNAMES> page 246

<@ARRAY>

Syntax

<@ARRAY [ROWS=*rows*] [COLS=*cols*] [VALUE=*textValue*] [CDELIM=*columnDelimString*] [RDELIM=*rowDelimString*]>

Description

Returns an array with a specified number of rows and columns.

This Meta Tag is usually used in conjunction with <@ASSIGN>. See the examples in this section.

The attributes ROWS and COLS optionally specify the number of rows and columns in the array, respectively. The optional attribute VALUE specifies a string used for initializing the array, formatted as array elements separated by CDELIM and RDELIM text.

ROWS and COLS must be specified if VALUE is not specified. VALUE must be specified if ROWS and COLS are not specified.

If all three of these attributes are specified, they must be in accord, or an error is generated. The following example would generate an error because the VALUE specifies three columns and two rows, which contradicts the ROWS and COLS attributes.

```
<@ARRAY ROWS=10 COLS=2 VALUE="a,b,c;d,e,f">
```

It is also invalid to specify a VALUE attribute with different numbers of columns in each row. The number of columns in each row must be the same, and must match the COLS value, if specified.

If the CDELIM and RDELIM attributes were specified as"," and ";", respectively, and the value string were specified as VALUE="1,2,3;4,5,6;7,8,9;a,b,c;" an array with the following structure would be created:

```
1 2 3
4 5 6
7 8 9
a b c
```

For more information, see "cDelim" on page 389 and rDelim <page $pagenum>.

If no values for the column or row delimiters are specified, then the values specified by the configuration variables cDelim and rDelim are used as defaults.

Working with Arrays

There are several Meta Tags available to manipulate arrays. One group of Meta Tags works on a single array. The <@DISTINCT> Meta Tag searches an array and displays only the unique or distinct rows. The <@FILTER> Meta Tag allows you to create a new array from an

existing array based on certain criteria. The <@SORT> Meta Tag allows you to sort an array. The <@DELROWS> Meta Tag allows you to remove rows from an array.

Another group of Meta Tags work on rows in more than one array. The <@INTERSECT> Meta Tag compares the rows in two arrays, then displays only the rows that appear in both arrays. The <@UNION> Meta Tag compares the rows in two arrays, then displays only the distinct (non-duplicated) rows from both arrays. The <@ADDROWS> Meta Tag combines the rows of two arrays without analysis.

Examples

```
<@ASSIGN NAME="array1" VALUE="<@ARRAY ROWS='6'
COLS='3'>">
```

This creates an array and assigns it to a variable.

```
<@ASSIGN NAME="initValue"
VALUE="1,2,3;4,5,6;7,8,9;a,b,c;d,e,f;g,h,i">

<@ASSIGN NAME="array2" VALUE="<@ARRAY ROWS='6'
COLS='3' VALUE=@@initValue CDELIM=',' RDELIM=';'>">

<@VAR NAME="array2">
```

This creates and initializes an array, assigns it to a variable, and prints it.

See Also

<@ADDROWS>	page 20
<@ASSIGN>	page 33
<@DELROWS>	page 103
<@DISTINCT>	page 105
<@FILTER>	page 148
<@INTERSECT>	page 171
<@SORT>	page 256
<@UNION>	page 277
<@VAR>	page 292

<@ASCII>

Syntax

```
<@ASCII CHAR=char>
```

Description

Returns the ASCII value of the first character of the string specified in the CHAR attribute.

Note Characters with ASCII codes above 127 return different values depending on the character encoding standard used by the operating system on the computer where TeraScript Server is running.

The attribute may be a literal value or a Meta Tag that returns a string.

Examples

```
<@ASCII CHAR="T"> or <@ASCII “T”>
```

This example returns “84”.

```
<@ASCII CHAR="<@POSTARG NAME=FirstName>">
```

This example returns the ASCII value of the first character of the FirstName field of the HTML form.

See Also

<@CHAR> page 60

<@ASSIGN>

Syntax

```
<@ASSIGN NAME=name VALUE=value [SCOPE=myscope]
[EXPIRES=timestamp] [PATH=path] [DOMAIN=domain]
[SECURE=true|false]>
```

Description

Assigns a value to a variable. If the specified variable does not yet exist, it is created.

For more information on variables see Working With Variables page 319.

The NAME attribute specifies the name of the variable to assign the value to. The following restrictions apply to the value specified in the NAME attribute:

- must start with a letter (a-z)
- may contain numbers (0-9), letters (a-z), and the underscore character "_". (Note that the use of the "." character in the NAME attribute has been deprecated in TeraScript 6.)
- must be no longer than 40 characters.

Variable names are case insensitive; for example, myVar is the same variable as MYVAR and MyVaR.

The value may be text, an array, email or DOM.

If the variable being assigned to exists and contains an array, this tag also lets you set the values of individual elements in that array. <@ASSIGN> can assign an array (or array section) to a variable, or to another array (or array section). Array assignments require that the source and target arrays (or array sections) have the same dimensions.

If you are assigning to an array variable element or section, the name includes the element or section specification specified within square brackets as [rownumber,colnumber], with an asterisk indicating all rows or all columns; for example, NAME=myArray[1,2] or NAME=myArray[*,3].

The VALUE attribute specifies the value to assign to the variable. If you are assigning to an array section, the value specified here must match the dimensions of the array variable specification in NAME.

Note You can add rows to an array, but not columns. For more information, see <@ADDROWS> <page $pagenum>. Resizing an array variable is not supported, but you may assign a new array (of any dimension) to an existing variable. Assigning subset shapes is not possible where such shapes cannot be described with the wildcard syntax "*".

Scope Attributes

Scoping is the method by which variables can be organized and disposed of in an orderly and convenient fashion. There are various levels of scoping, each of which has an appropriate purpose:

For more information, see "Configuration Variables" on page 373.

- **System Scope** contains any variables that are general to all users. This scope contains only TeraScript Server configuration variables. To use this scope, specify SCOPE=system or SCOPE=sys.

For more information, see "domainScopeKey" on page 412 .

- **Domain Scope** contains variables that users can share if they are accessing a particular TeraScript application file from a specified TeraScript domain. TeraScript domains are specified in a domain configuration file, or default to the domain name (base URL or IP address) of the path to the TeraScript application file. This scope is defined by setting the system configuration variable domainScopeKey appropriately; that is, setting it to a value that can differentiate such users. By default, this is <@DOMAIN>, which returns the value of the current TeraScript domain.To use this scope, specify SCOPE=domain.
- **Application Scope** contains variables that are shared across TeraScript applications. TeraScript applications are defined by TeraScript users in an application configuration file. To use this scope, specify SCOPE=application or SCOPE=app.
- **User Scope** contains variables that a user defines and expects to be able to access from many application files or invocations of single application files. To use this scope, specify SCOPE=user or SCOPE=usr.

- **Request Scope** contains variables that should be unique to every invocation of any application file. For example, this scope could be used for temporary variables that reformat output from a search action. All variables of this scope are removed

when the application file concludes execution. To use this scope, specify SCOPE=request, or SCOPE=doc.

Note Applications which refer to the Request by its old name, the Local scope, in version 6.2 will have a warning logged during execution. Developers are encouraged to cease using the Local scope name to prevent conflicts in future versions of the software.

- **Instance Scope** contains variables that are valid in an instance of a TeraScript class file. These variables can be shared across methods called on a TeraScript class file, if the methods are called on the same instance. To use this scope, specify SCOPE=instance.
- **Method Scope** contains variables that should be unique to a method of a TeraScript class file. To use this scope, specify SCOPE=method.
- **Cookie Scope** contains variables that are sent to the user's Web browser as cookies (that is, a small text file kept by the Web browser for a specified amount of time). To use this scope specify SCOPE=cookie.
- **Custom Scope** is user-specified. It is outside of the scope search hierarchy.

For more information on "Scoping" see Understanding Scopepage 321.

If this attribute is omitted, the following steps are taken to determine the scope in which the assignment takes place:

TeraScript searches for the variable in request, user, domain and system scope, in that order. As soon as the variable by the NAME specified is found, the search stops, and the VALUE is assigned to that variable.

A new variable is created in the default scope if the variable is not found. The default scope is normally REQUEST, but can be changed by setting the defaultScope configuration variable in the terascript.ini file.

Cookie Attributes

The EXPIRES, PATH, DOMAIN, and SECURE attributes are only valid when SCOPE=COOKIE.

For more information on Cookie Scope, seeFor more information, see "Cookie Scope" on page 323

If the EXPIRES attribute is omitted, the cookie expires when the user quits their Web browser. This is the default cookie behavior as described in the cookie specifications. Otherwise, a GMT timestamp must be specified in the following format:

```
Wdy, DD-Mon-YY HH:MM:SS GMT
```

The following EXPIRES attribute is a combination of Meta Tags that specifies a GMT date in the correct format based on the current timestamp plus one week (604,800 seconds):

```
EXPIRES=<@TOGMT TS=<@SECSTOTS SECS='<@CALC
EXPR="<@TSTOSECS
TS=<@CURRENTTIMESTAMP>>+604800">'>
FORMAT="datetime:http">
```

For more information, see "<@CURRENTTIMESTAMP>" on page 88, "<@TSTOSECS>" on page 275, and <@TOGMT> <page $pagenum>.

If the DOMAIN attribute is omitted, the Domain value is omitted from the Set-Cookie line, causing the cookie to be valid for the current server. Otherwise you can specify any domain string up to 63 characters. .example.com, for example, would cause the cookie to be sent back to www.example.com, demo.example.com, sales.example.com, and so on.

In the PATH attribute, server root (/) specifies that the cookie be sent for all paths within the specified domain. You can specify a path string up to 63 characters. For example, /TeraScript/ would cause the cookie to be sent back only for URLs below the TeraScript folder. If no PATH is specified, the default is server root.

The SECURE attribute specifies whether a secure connection is required for client send. Possible values are TRUE (enabled) or FALSE (disabled). This option sets the Secure value of the Set-Cookie line. If the value is set to TRUE, then the cookie is sent back by the Web browser only if a secure connection is being made. The default is FALSE, which is used if no secure attribute is found.

Examples

```
<@ASSIGN NAME="foo" VALUE="123456" SCOPE="user">
```

This example assigns the value "123456" to the variable foo in user scope.

```
<@ASSIGN NAME="foo2" VALUE="abcdef">
```

This example either assigns the value "123456" to the variable foo2 in request, user, application, domain or system scope, depending on the first instance of foo2 that TeraScript Server encounters; or, if it does not exist, it creates a new variable called foo2 in default scope and assigns the value "123456" to it.

```
<@ASSIGN NAME=fred SCOPE=cookie VALUE="You were
here." EXPIRES="<@TOGMT TS=<@SECSTOTS SECS='<@CALC
EXPR="<@TSTOSECS TS=<@CURRENTTIMESTAMP>>+604800">'>
FORMAT="datetime:http">">
```

This example sends a cookie named fred that is valid for the current server and path, has the value "You were here." and expires in one week.

```
<@ASSIGN NAME="foo3" SCOPE="user" VALUE="<@ARRAY
ROWS=5 COLS=3>">
```

This example assigns an empty array of five rows and three columns to the user variable `foo3`.

```
<@ASSIGN NAME="foo4" SCOPE="user"
VALUE="<@POSTARGNAMES>">
```

This example assigns the evaluated value of the Meta Tag `<@POSTARGNAMES>` (an array) to the user variable `foo4`.

```
<@ASSIGN NAME="initValue"
VALUE="1,2,3;4,5,6;7,8,9;a,b,c;d,e,f;g,h,i">
<@ASSIGN NAME="array2" VALUE="<@ARRAY ROWS='5'
COLS='3' VALUE=@@initValue CDELIM=','
RDELIM=';'>">
<@ASSIGN NAME="foo5" SCOPE="user"
VALUE="@@array2[*,2]">
```

This example creates a 5 row by 3 column array variable named `array2`, initializes it with the content of `initValue`, and then creates a new one-column array variable (`foo5`), containing all the values in column 2 of `array2`.

```
<@ASSIGN NAME="orders[1,*]" VALUE="@@myOrder"
SCOPE="user">
```

This example puts the single-row array stored in the `myOrder` variable into the first row of the `orders` user variable, replacing the existing values. This assignment generates an error if `myOrder` is not an array, contains more than one row, or does not contain the same number of columns as the `orders` array.

```
<@ASSIGN NAME="zips" VALUE="@@orders[*,4]"
SCOPE="request">
```

Assigns to the request variable `zips` a one-column array of all the values from column 4 of the `orders` array.

```
<@ASSIGN NAME="curr_cust" VALUE="@@orders[1,1]">
```

Assigns the value from the first cell in the first row of the `orders` array to the `curr_cust` variable, using default scoping rules.

```
<@ASSIGN NAME="race_results[*,3]" VALUE="<@VAR
NAME='new_results[*,1]'>">
```

Copies the values from column 1 of the `new_results` array to the third column of the `race_results` array. Both arrays must contain the same number of rows, or an error occurs.

See Also

`<@ARRAY>`	page 30
`<@PURGE>`	page 225

`<@VAR>` page 292
`<@DEFINE>` page 100
`variableTimeout` page 478
`Working With Variables` page 319

<@BIND>

Syntax

```
<@BIND NAME=varname [DATATYPE=datatype] [SCOPE=scope]
[BINDTYPE=bindtype] [PRECISION=number] [SCALE=number]
[BINDNAME=bindname]>
```

Description

The <@BIND> Meta Tag is used to pass a value in the Direct DBMS action using the parameter binding capabilities of ODBC or OCI. This Meta Tag instructs TeraScript Server to generate the appropriate binding calls based on the assigned data source type. Binding is useful for passing values to, and retrieving values from, stored procedures.

The NAME attribute is the name of a TeraScript variable to be used for parameter binding. The SCOPE attribute is an optional attribute defining the scope of the variable named in the NAME attribute.

The BINDTYPE attribute can have one of the following values:

- IN. Input parameter only. The execution of the SQL never affects the contents of the TeraScript variable. This is the default value for the BINDTYPE attribute.
- IN/OUT. The parameter is used by the SQL (normally, a stored procedure call) for both input and output.
- OUT. Output parameter only. Output parameters are set by the stored procedure being called. The value before execution of the <@BIND> tag is irrelevant to the execution.

The DATATYPE attribute defines how the contents of the TeraScript variable will be interpreted when actual DBMS binding is performed. Valid datatypes are INTEGER, VARCHAR, CHAR, DATE, TIME, TIMESTAMP, REAL, FLOAT, NUMERIC, and DECIMAL.

If the DATATYPE is set to TIMESTAMP, the timestamp text for ODBC data sources should be in the following form:

```
YYYY-MM-DD HH:MI:SS[.FFFFFF].
```

For OCI data sources, the default format is the one for the current session; for example, DD-MONTH-YY.

The PRECISION attribute is the size of the bound parameter in bytes. The default value for this attribute is that of the corresponding datatype, that is, 255 for VARCHAR and CHAR, 10 for INTEGER, and so on, and needs to be changed only if the corresponding bound DBMS parameter has the precision less than the default value.

The SCALE attribute is the number of the decimal digits after the floating point for numeric data types. The value of the SCALE attribute is ignored for textual parameters.

The following table gives the default values for PRECISION and SCALE for various values of DATATYPE:

DATATYPE	Default PRECISION	Default SCALE
INTEGER	10	0
FLOAT, REAL, NUMERIC, DECIMAL	15	2
VARCHAR, CHAR	255	-
DATE, TIME, TIMESTAMP	-	-

In case of an input-only (IN) parameter, PRECISION has no effect and is overridden by the actual parameter length. For IN/OUT text parameters, the PRECISION attribute defines the maximum length of the text returned by the DBMS.

The BINDNAME attribute is an optional name used as a bound parameter placeholder when the SQL statement is processed. This attribute is used only if the underlying DBMS driver supports this functionality. Currently, only OCI supports named parameters.

Since both OCI and ODBC provide implicit type castings, handling integer, float and varchar columns allow you to bind virtually any data necessary, with the exception of VARBIN columns, because the length is currently limited to 32767.

The use of <@BIND> with TYPE=IN is valuable in that the contents of the bound parameter can contain characters such as quotes, commas, and control characters that do not affect the execution of the stored procedure: there is no need to quote a bound parameter.

Caution If an error occurs that prevents the successful completion of an action using bound output variables, the values in those variables are undetermined and no assurance is given that they have or have not been modified. Furthermore, a Transaction Rollback or Commit action issued to the data source does not affect the values in variables previously bound within a Direct DBMS action involved in the transaction.

Limitations

Since TeraScript Server does not parse the SQL inside a Direct DBMS action other than to perform the above substitutions, the OUT parameters of one stored procedure call cannot be used as input to another stored procedure call within the same action.

For example, if the same Direct DBMS action specified above also called another function:

```
CALL UpdatePersonalExpenses(<@VAR premium>);
```

The value for premium that was calculated by the `CalculateMortgagePremium()` function would not be available for this stored procedure.

Caution Only ODBC v2.0 or above drivers support `IN/OUT` parameters through the `SQLBindParameter()` call.If `BINDTYPE=IN/OUT` is specified for the `TIMESTAMP` datatype, the Oracle ODBC driver returns an error message. The bind type IN works correctly.

Example

```
create procedure sp_testproc
@param1 varchar(64) output,
@param2 integer output,
@param3 float output,
@param4 numeric(10,2) output,
@param5 datetime output
as
select @param1 = @param1 + ': param1'
select @param2 = @param2 + 2
select @param3 = @param3 + 3.3
select @param4 = @param4 + 4.4
select @param5 = CONVERT(datetime, getdate())
```

This creates a SQL Server stored procedure `sp_testproc`.

```
{CALL sp_testproc(
<@BIND NAME=Param1 BINDTYPE=IN/OUT
BINDNAME=StringParam PRECISION=32>,
<@BIND NAME=Param2 BINDTYPE=IN/OUT
DATATYPE=INTEGER>,
<@BIND NAME=Param3 BINDTYPE=IN/OUT DATATYPE=FLOAT
SCALE=3>,
<@BIND NAME=Param4 BINDTYPE=IN/OUT DATATYPE=DECIMAL
PRECISION=6 SCALE=2>,
<@BIND NAME=Param5 BINDTYPE=IN/OUT
DATATYPE=TIMESTAMP>
)}
```

This calls the preceding stored procedure `sp_testproc`.

<@BREAK>

Description

Terminates execution of a <@COLS>, <@ROWS>, <@FOR>, or <@OBJECTS> block. <@BREAK> causes execution to continue to the HTML following the current loop's close tag; outside of a loop, it does nothing. This tag has no attributes. This tag does not affect loops initiated with For Loop or While Loop actions.

This tag is generally used with an <@IF> tag to terminate a loop when some condition is met. Be careful to handle nested loops properly: only the innermost loop's processing is affected by the break.

Example

```
<@ASSIGN NAME="running_total" VALUE="0">
<@ROWS>
Here are the values from record <@CURROW> of the
results:<P>
<STRONG>Company Name:</STRONG> <@COLUMN
NAME="company.name"><BR>
<STRONG>Balance</STRONG> $<@COLUMN
NAME="company.balance"><BR>
Running total: $<@NEXTVAL NAME="running_total"
STEP='<@COLUMN NAME="company.balance">'>
<@IF EXPR="@@running_total>=1000" TRUE="<@BREAK>">
</@ROWS>
Running total of balance has reached $1000. End of
records.
```

This example returns records until the accumulated total of all the company.balance columns reaches or exceeds 1000.

See Also

<@COLS> </@COLS>	page 75
<@CONTINUE>	page 82
<@EXIT>	page 147
<@FOR> </@FOR>	page 151
<@OBJECTS></@OBJECTS>	page 217
<@ROWS> </@ROWS>	page 239

<@CALC>

Syntax

<@CALC EXPR=*expr* [PRECISION=*precision*] [FORMAT=*format*] [ENCODING=*encoding*]>

Description

Returns the result of the calculation specified in EXPR.

Basic Functionality

The expression may contain numbers (including numbers in scientific notation); any of six arithmetic operations—multiplication (*), division (/), modulo (%), power (^), addition (+), and subtraction (-); parentheses for controlling the order of operations; mathematical functions; string functions; logical operations; comparison operations; calculation variables (A–Z); and sub-expressions.

Note Do not confuse calculation variables with configuration variables or other TeraScript variables. They are only applicable to <@CALC> and do not work with <@ASSIGN> or <@VAR>.

If the expression contains any spaces—except for spaces within embedded Meta Tags—it must be quoted.

The optional PRECISION attribute is an integer that controls the number of decimal places displayed in the result. The *default precision* is the maximum required for accuracy of the result.

If an error is encountered while the expression is parsed or computed, the computation is halted and the tag is substituted with relevant error information.

Examples

```
<@CALC EXPR="3+7">
```

This tag returns “10”.

```
<@CALC EXPR="<@POSTARG NAME='calculateThis'>"
PRECISION="4">
```

This evaluates the contents of the form field specified—calculateThis—to four decimal places of precision. If the field contained “8*9+8/2”, for example, the tag would evaluate to “76.0000”.

Numbers

A valid number is a sequence of digits, optionally preceded or trailed by a currency sign (default "$", otherwise set by the configuration variable `currencyChar`), with any number of thousand separator characters, an optional decimal point, and an exponentiation part. As well, an empty variable or empty string evaluates to zero.

Numbers can be used with any operators and functions, even with the string specific function *len*, which returns the length of the number converted to a string.

When a number is used in logical expression, any non-zero number is considered *true*, and zero is considered *false*.

Logical expressions themselves return "1" if they are *true* or "0" if they are *false*.

Two symbolic constants, `true` and `false`, which evaluate to "1" and "0", respectively, are provided for convenience.

An empty string evaluates to zero for the purposes of calculation. That is, if the variable `foo` is empty, the following operations are valid:

```
<@CALC '@@foo + 1'>        OK, returns 1
<@CALC '"" + 1'>           OK, returns 1
<@CALC 'mean(@@foo 1)'>    OK, returns 0.5
```

The thousand separator set to space

A special case occurs when the thousand separator is set to a space. A number containing a space can be processed if it is a result of a tag evaluation; however, a number literal must be quoted if it includes spaces.

For example:

```
<@ASSIGN NAME=fred VALUE="1 000 000">

<@CALC "@@fred / 100">     Ok, returns 10000.0
<@CALC "@@fred > '1 000'"> Ok, returns 1.0
<@CALC "@@fred > 1 000">   Error
```

For more information, see "currencyChar" on page 396, `decimalChar <page $pagenum>`, `DBDecimalChar <page $pagenum>`, and `thousandsChar <page $pagenum>`.

The thousands separator, currency sign, and other numerical formats are set by TeraScript configuration variables. They can be set in various scopes.

Array evaluation

<@CALC> treats array references using non-array-specific operators and functions as a numerical value returning the number of rows in the array.

This provides an easy way to verify whether an array is empty or contains a certain value. For example, you can test for the existence of an array variable with `<@CALC EXPR="@@array_variable > 0" TRUE="Yes!" FALSE="No such variable.">`.

For example:

The variable `fred` contains the following array:

1	2
3	4

The variable `barney` contains the following array:

1	2
5	6
7	8

`<@CALC @@fred>` returns 2.

`<@CALC @@barney>` returns 3.

`<@IF EXPR="@@fred > @@barney" TRUE="true!" FALSE="alas">` returns “alas”.

Hexadecimal, Octal and Binary Numbers

The calculator can accept hexadecimal, octal, and binary numbers. The *num* function converts strings representing hexadecimal, octal and binary numbers to decimal numbers, and the result of the conversion can be used anywhere where a number is used. The following table specifies the conversion rules.

Note If a decimal number is passed to this function, it either yields an error or an incorrect result.

Prefix	Valid Symbols	Converted As	Examples
0x	0123456789abcdef	Hexadecimal	num (*0xff*) num (*0x0123f3a4*)

Prefix	Valid Symbols	Converted As	Examples
0	01234567	Octal	num (*0123456*) num (*0120235*)
None	01	Binary	num (*1011110010100*) num (*111*)

For example, all the following expressions generate errors:

```
num(0x123fga)      ERROR: letter g is invalid
num(012380)        ERROR: digit 8 is invalid
num(123)           ERROR: digits 2 and 3 are invalid
```

Strings

Any TeraScript Meta Tag that does not evaluate to a valid number or array reference is considered a string. No additional quoting is required. There is a single exception to this rule, further explained in `Meta Tag Evaluation <page $pagenum>`.

Strings can be used only in comparison operations, *contains* clauses or as arguments to the *len* function. A string literal—that is, a string, directly included in the expression—must be enclosed in single quotes if it contains spaces, special characters or starts with a digit.

For more information, see "Calculation Variables" on page 47.

Note Single letters must always be enclosed in quotes in string operations so that they are treated as letters, and not as calculation variables.

```
<@ASSIGN NAME=name VALUE="John Lennon">
<@CALC EXPR="@@name=John"> false
<@CALC EXPR="@@name=John Lennon"> ERROR
<@CALC EXPR="@@name='John Lennon'"> true
<@CALC EXPR="@@name='John*'"> true

<@ASSIGN NAME=name VALUE="John's trousers">
<@CALC EXPR="@@name=John*" true
<@CALC EXPR="@@name='John\'s trousers'"> true
<@CALC EXPR="@@name='John's'"> ERROR

<@ASSIGN NAME=dir VALUE="C:\test">
<@CALC EXPR="@@dir='C:\test'"> false
<@CALC EXPR="@@dir='C:\\test'"> true
```

These examples show string comparisons. If a string literal contains a single quote or a backslash, it must be escaped with a backslash.

When a string is encountered on one side of the comparison operation, the other operand is forced to a string, too. For example:

```
2.15 <='abba'
'123.456.78.12'=@@ip_address
```

Function *len* returns the length of the string, so the result of this operation can be used anywhere a number can be used. Strings can not be assigned to calculation variables.

For example, these are valid expressions:

```
ABBA='BLACK SABBATH' false
len(JOHN LENNON) + len(FREDDY MERCURY) - 5 > 0 true
```

but these are not:

```
a :=ABBA        ERROR: cannot assign string
FREDDY < 0      ERROR: cannot compare string and number
```

and this tag returns true although you may expect it to return false:

```
<@CALC EXPR="a=b">
```

Note A single letter on both sides of the comparison operator evaluates to a calculation variable, meaning a number comparison is performed.

String comparisons using <@CALC> are case insensitive.

Calculation Variables

A calculation variable is a single case-insensitive letter (A–Z) that can be assigned a numeric value and used in subsequent operations. You can write small programs inside the tag with calculation variables and statement separators, or put a program in a separate file and use <@INCLUDE> to calculate the result.

Single letters must always be enclosed in quotes in string operations so that they are treated as letters, and not as calculation variables. For example:

For more information, see "beginswith" on page 49.

<@CALC EXPR="Henry beginswith 'H'"> evaluates the string "Henry" to see if it begins with the string "H" (case-insensitive).

<@CALC EXPR="1234 beginswith H"> evaluates "1234" to see if it begins with the value specified in the calculation variable H (number-to-string conversions are performed).

The following table shows predefined calculation variables. You may use these values in your programs, or have any of these calculation variables reassigned with any other value.

Variable	Meaning	Value
G	**(3 - sqrt(5))/2**, the golden ratio.	0.381966011250105
E	**e**, the base of natural logarithms.	2.718281828459045
L	**$\log_{10}$(e)**, the ratio between natural and decimal logarithms.	0.434294481903252
P	**pi**, the circumference to diameter ratio of a circle.	3.141592653589793
Q	**sqrt(2)**, the square root of 2.	1.414213562373095
I	Has a meaning only inside *foreach* expression.	Current row index
J	Has a meaning only inside *foreach* expression.	Current column index
X	Has a meaning only inside *foreach* expression.	Current array element index

Operators

The following table shows the operators listed in order of increasing precedence. Operators having the same precedence, for example, plus and minus, are not separated by a rule.

Note The `beginswith` operator should be used instead of a trailing asterisk as a wildcard in comparisons.The use of asterisks as wildcards is deprecated and will be removed in a future release.

Operator	Meaning and Return Value	Usage
;	Sub-statement separator, returns the value of the last statement.	*statement* ; *statement*
:=	Assignment operator, assigns the value of the expression to the calculation variable, and returns that value.	*variable* := *expression*
|| OR	Logical OR, returns 1 if any of the expressions is evaluated to a non-zero value, or 0 otherwise.	*expr* || *expr* *expr* OR *expr*
&& AND	Logical AND, returns 1 if both of the expressions are evaluated to non-zero values, or 0 otherwise.	*expr* && *expr* *expr* AND *expr*

Operator	Meaning and Return Value	Usage
<	Numeric or string LESS. Returns 1 if left operand is greater than right one, or 0 otherwise.	*expr* < *expr* *string* < *string*
>	Numeric or string GREATER. Returns 1 if left operand is greater than right one, or 0 otherwise.	*expr* > *expr* *string* > *string*
<=	Numeric or string LESS OR EQUAL. Returns 1 if left operand is less than or equal to right one, or 0 otherwise.	*expr* <= *expr* *string* <= *string*
>=	Numeric or string GREATER OR EQUAL. Returns 1 if left operand is greater than or equal to right one, or 0 otherwise.	*expr* >= *expr* *string* >= *string*
=	numeric or string EQUAL. Returns 1 if left operand is equal to right one, or 0 otherwise.	*expr* = *expr* *string* = *string*
!=	Numeric or string NOT EQUAL. Returns 1 if left operand is not equal to right one, or 0 otherwise.	*expr* != *expr* *string* != *string*
? :	Ternary comparison. Evaluates to *expr1* if condition is true, or to *expr2* otherwise.	(*cond*) ? *expr1*: *expr2*
contains	Containment. Returns true if specified string or number is contained in the array.	*array* contains *string* *array* contains *number*
contains	Occurrence. Returns true if specified string or number is a substring of the source string.	*source_string* contains *string* *source_string* contains *number*
beginswith	Occurrence. Returns true if specified string or number begins the source string. (Case-insensitive.)	*source_string* beginswith *string* *source_string* beginswith *number*
endswith	Occurrence. Returns true if specified string or number ends the source string. (Case-insensitive.)	*source_string* endswith *string* *source_string* endswith *number*
+	Addition. Returns the sum of the expressions.	*expr* + *expr*
–	Subtraction. Returns the difference of the expressions.	*expr* – *expr*

Operator	Meaning and Return Value	Usage
*	Multiplication. Returns the product of the expressions.	*expr * expr*
/	Division. Returns the quotient of the expr1 divided by the expr2.	*expr1 / expr2*
%	Modulo. Returns the remainder of expr1 divided by expr2.	*expr1 % expr2*
^	Power. Returns expr1 raised to expr2 power.	*expr1 ^ expr2*
–	Unary minus. Returns the negation of the expression.	*– expr*
+	Unary plus. Returns the expression itself.	*+ expr*
! NOT	Logical NOT. Returns 0 if the value of the expression is not 0, or 1 otherwise.	*! expr* NOT expr

Built-in Functions

Each built-in function expects either a single numeric argument, or a space-separated list of mixed numeric and array arguments, or a string. It is an error to specify an argument of the wrong type to a function. If an array, specified as an argument to a function, contains non-numeric elements, these elements are ignored without any error diagnostics.

The following tables list all built-in functions.

TABLE 1. Numeric functions of the form *func(expr)*

Function	Meaning and Return Value	Arguments and Usage
abs	**|x|**, the absolute value of the expression	abs(*expr*)
acos	$\boldsymbol{cos^{-1}(x)}$, the arccosine of the expression, returned in radians	acos(*expr*)
asin	$\boldsymbol{sin^{-1}(x)}$, the arcsine of the expression, returned in radians	asin(*expr*)
atan	$\boldsymbol{tan^{-1}(x)}$, the arctangent of the expression, returned in radians	atan(*expr*)
ceil	expression rounded to the closest integer greater than or equal to the expression	ceil(*expr*)
cos	***cos(x)***, the cosine of the expression, specified in radians	cos(*expr*)
exp	$\boldsymbol{e^x}$, the exponentiation of the expression	exp(*expr*)

Function	Meaning and Return Value	Arguments and Usage
fac	***x!*** (or ***1*2*3*...*x***) factorial of the expression	fac(*expr*)
floor	expression rounded to the closest integer less than the expression	floor(*expr*)
log	***ln(x)*** (or $\boldsymbol{log_e(x)}$), the natural logarithm of the expression	log(*expr*)
log10	$\boldsymbol{log_{10}(x)}$, the decimal logarithm of the expression	log10(*expr*)
sin	***sin(x)***, the sine of the expression, specified in radians	sin(*expr*)
sqrt	***sqrt(x)*** (or $\boldsymbol{x^{1/2}}$), the square root of the expression	sqrt(*expr*)
tan	***tan(x)***, the tangent of the expression, specified in radians	tan(*expr*)

TABLE 2. String functions of the form *func(string)*

Function	Meaning and Return Value	Arguments and Usage
len	returns the length of the string enclosed in parentheses	len(*text*)
num	converts a string, representing a hexadecimal, octal, or binary number into a number	num(*text*)

TABLE 3. Array functions of the form *func(expr|array expr|array)*

Function	Meaning and Return Value	Arguments and Usage
max	$\boldsymbol{max(A_1, A_2,...A_n)}$. returns the largest element	max(*expr expr ...*)
min	$\boldsymbol{min(A_1, A_2,...A_n)}$. returns the smallest element	min(*expr expr ...*)
sum	$\boldsymbol{A_1+A_2+...+A_n}$. returns the sum of the elements	sum(*expr expr...*)
prod	$\boldsymbol{A_1*A_2*...*A_n}$. returns the product of the elements	prod(*expr expr...*)

Function	Meaning and Return Value	Arguments and Usage
mean	$A_{mean}=(A_1+A_2+...+A_n)/n$. returns the mean of the elements	mean(*expr expr...*)
var	$A_{var}=((A_1-A_{mean})^2+((A_2-A_{mean})^2+...+((A_n-A_{mean})^2)/(n-1)$ returns the (squared) variance of the elements	var(*expr expr...*)

Array Operators

Contains Operator

The *contains* operator has the following syntax:

```
<@VAR NAME="array"> contains number or string
```

This operator checks if the specified number or string is contained in the array. The string should be enclosed in quotes, if it contains any non-alphanumeric characters. The operator returns "1" if the element is found, or "0" otherwise.

```
<@IF EXPR="<@VAR NAME=CDs> contains Queen" TRUE=Cool
FALSE="Too bad">
```

In this example expression, which uses the <@IF> Meta Tag, returns "Cool" if "Queen" is found in the CDs array, and "Too Bad" if it is not.

Foreach Operator

The *foreach* operator has the following syntax:

```
<@VAR array> foreach {statement; ...}
```

For more information, see "<@ARRAY>" on page 30.

This operator steps through the elements of an array and it assigns

- the value of the elements to the variable "X**"**
- the current row number to the variable "I"
- and the current column number to the variable "J"

For more information, see "<@ASSIGN>" on page 33.

and it executes the statements inside the braces "{ }"for each element. All non-numeric elements are interpreted as zeroes.

The operator returns the last calculated value of the expression.

The values of "X, I, J" are restored upon the exit from the foreach operator. For example, if array CDs is initialized as follows:

```
<@ASSIGN NAME="CDinitValue" VALUE="AC/
DC,Scorpions,Deep Purple,Black
Sabbath,Queen;19.50,22.50,22.50,17.90,29.00">
```

```
<@ASSIGN NAME="CDs" VALUE="<@ARRAY ROWS='2'
COLS='5' VALUE=@@CDinitValue CDELIM=','
RDELIM=';'>">
```

then the following program prints the name of the most expensive CD:

```
<@VAR NAME=CDs[1,<@CALC "t :=1; p :=0.0;
<@VAR NAME=CDs> foreach
{ t :=(p < x)? j: t; p :=(p < x)? x: p; }; t">]>
```

Meta Tag Evaluation

There are two special cases when a Meta Tag is not treated as a string. Consider the following two examples:

```
<@CALC EXPR="<@POSTARG NAME=prog>">
<@CALC EXPR="<@INCLUDE FILE=myprog>">
```

If the post argument *prog* contains an expression submitted by a user, or the file *myprog* contains an expression to be calculated, one would expect <@CALC> to produce the result of the calculation. The rule is, if the expression contains a single Meta Tag, such an expression is fully evaluated by the calculator, rather than treated as a string.

Ordering of Operation Evaluation With Parentheses

Parentheses can be used to order the evaluation of expressions that otherwise are evaluated in the order specified in the Operators table (page 48). For example:

```
<@CALC EXPR= "7*3+2">
```

This example evaluates to “23”.

```
<@CALC EXPR= "7*(3+2)">
```

This example evaluates to “35”.

A more complex example can be constructed using different operators and nested parentheses:

```
<@CALC EXPR="(<@ARG _function> = 'detail') and
((len(<@ARG id>) != 0 and <@ARG mode>='abs')
or (<@ARG mode>='next' or <@ARG mode>='prev'">))>
```

This tag evaluates to “1” (true) if the `_function` argument is equal to “detail” *and* any one of the following conditions are met:

- `id` arg is not empty *and* the `mode` arg is “abs”
- `mode` argument is “next”

- mode argument is “prev”.

See Also

<@ARRAY>	page 30
<@ELSEIF>	page 161
<@ELSEIFEMPTY>	page 161
<@ELSEIFEQUAL>	page 161
Encoding Attribute	page 8
<@FORMAT>	page 153
Format Attribute	page 11
<@IF>, <@ELSE>	page 161
<@VAR>	page 292

<@CALLMETHOD>

Syntax

<@CALLMETHOD OBJECT=*variable* METHOD=*method&parameters* [SCOPE=*scope*] [METHODTYPE=*get*|*set*|*invoke*] [PARAMTYPES=*paramtypes*]>

Description

This Meta Tag calls a specified method of an object.

The OBJECT attribute defines the name of a variable containing an object instance. The optional SCOPE attribute defines the scope of the variable.

The METHOD attribute defines the name of the method to call and the parameters. The METHOD attribute is of the following form:

```
methodname(param1, param2, ...)
```

For COM and JavaBean methods, the method name is case-sensitive.

A parameter must be quoted (single or double quotes) if it contains a comma or has significant spaces at the beginning or end, but otherwise quoting is unnecessary. Literal quotes in parameter values must be specified using the <@SQ> and <@DQ> Meta Tags. Initial and trailing spaces outside of quotes are ignored.

Overloaded Methods and PARAMTYPES

An overloaded method occurs when an object has more than one method with the same name. These methods have different parameter lists.

The PARAMTYPES attribute must be specified for overloaded methods. This attribute is specified as a comma-delimited list of data types which have a one-to-one correspondence with the parameters specified in the METHOD attribute. If the attribute is not specified for overloaded methods, the method call may fail because TeraScript Server does not know which method of the object to call (it chooses the first one).

Type Conversion

At execution time, if the PARAMTYPES attribute is not specified, TeraScript Server introspects the named method to determine the data types of its parameters and does the necessary conversion.

Passing Parameters "By Reference"

For more information, see "<@VARPARAM>" on page 301.

Values from an out or in/out parameter are obtainable by binding the parameter to a TeraScript variable using the <@VARPARAM> Meta Tag. Object variables and arrays may be passed with <@VARPARAM> as well. This is equivalent to using the Variable option in the Format column of the Call Method action parameter.

Note The <@VARPARAM> Meta Tag *must* be used with parameters of type Variant (COM-only).

The METHODTYPE attribute specifies one of GET, SET, or (the default) INVOKE. The METHODTYPE attribute must be specified for getter and setter methods in order to prevent ambiguity of method names (a standard method called GetProperty would, at least with COM, conflict with the getter for the property property).

Meta Tags are allowed in all attribute values. The return value of the tag is the return value of the method call.

Example

```
<@CALLMETHOD OBJECT=myObject
METHOD='foo(1,"test")'>
```

Calls the method foo with the specified parameters on the myObject object instance variable.

```
<@CALLMETHOD OBJECT=myCOMObject
METHOD="ContentID()" METHODTYPE=GET>
```

This example gets the value of the ContentID attribute of the myComObject instance. The GET is omitted from the name which appears in TeraScript Editor's Attributes folder for that object.

Examples of METHOD specifications using <@VARPARAM>:

```
<@CALLMETHOD OBJECT=myVar METHOD=foo(<@VARPARAM
NAME=myFirstParam SCOPE=request>, secondparam)>
```

The value for the first parameter comes directly from the myFirstParam request variable. After execution, the variable contains the output for that parameter.

```
<@CALLMETHOD OBJECT=myVar METHOD=foo(<@VARPARAM
NAME=user$myVar DATATYPE=LONG>)
```

If the first parameter of foo expects a Variant, this METHOD specification causes the data in myVar to be passed as a long.

See Also

<@CREATEOBJECT> page 83
<@GETPARAM> page 154

<@NUMOBJECTS>	page 214
<@OBJECTAT>	page 216
<@OBJECTS></@OBJECTS>	page 217
<@SETPARAM>	page 253
<@VARPARAM>	page 301

<@CGI>

Syntax

```
<@CGI [ENCODING=encoding]
```

Description

Returns the full path and name of the TeraScript CGI.

With server plug-in/module versions of the TeraScript client, this Meta Tag returns nothing.

Use this Meta Tag when creating embedded links to other TeraScript application files. Doing so ensures that the links work regardless of the Web server setup, or on which platform you are running TeraScript Server.

Note This Meta Tag is often used to create URLs, for example, in the HREF attribute of an anchor tag in HTML. To make sure that the Meta Tag returns a properly encoded value, you can use:

```
<@CGI ENCODING="URL">
```

Examples

```
<A HREF="<@CGI>/custlist.taf">List Customers</A>
```

This provides a link to an application file named `custlist.taf`.

With the CGI in the `cgi-bin` directory, the example returns `/cgi-bin/wcgi.exe/custlist.taf`. If you are running TeraScript with one of the server plug-ins, it returns `/custlist.taf`.

```
<A HREF="<@CGI>/more/cust_add.taf">Add Customer</A>
```

This links to the `cust_add.taf` application file located in a directory named `more` in the root directory of the Web server.

See Also

`<@APPFILE>`	page 22
`<@APPFILEPATH>`	page 24
`Encoding Attribute`	page 8
`<@MAKEPATH>`	page 202

<@CGIPARAM>

This tag was removed in the release of Witango 5.5 and was replaced by <@HTTPATTRIBUTE>.

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

See Also

<@HTTPATTRIBUTE> page 156

<@CHAR>

Syntax

<@CHAR CODE=*number* [ENCODING=*encoding*]>

Description

Returns the character that has the ASCII value *number*.

This Meta Tag is especially useful for specifying non-printing characters, such as linefeeds (<@CHAR CODE=10>), carriage returns (<@CHAR CODE=13>), and tabs (<@CHAR CODE=9>). Valid values for the number attribute are 1 through 254.

Note Numbers above 127 return different characters depending on the character encoding standard used by the operating system on the computer where TeraScript Server is running.

The number attribute may be a literal value or a Meta Tag that returns a number.

Examples

```
<@CHAR CODE=84> or <@CHAR “84”>
```

This example returns “T”.

```
<@CHAR CODE=<@POSTARG NAME=charCode>>
```

This example returns the character that corresponds to the value of the contents of the charCode form field entered by the user.

See Also

<@ASCII> page 32
<@CRLF> page 85
<@DQ>, <@SQ> page 120
Encoding Attribute page 8

<@CHOICELIST>

Syntax

```
<@CHOICELIST NAME=inputname TYPE=select|radio
OPTIONS=optionsarray [SIZE=size] [MULTIPLE=yes|no]
[CLASS=classname] [STYLE=stylename] [onBlur=script]
[onClick=script] [onFocus=script] [VALUES=valuesarray]
[SELECTED=selectedarray] [SELECTEXTRAS=selectattributes]
[OPTIONEXTRAS=optionattributes] [TABLEEXTRAS=tableattributes]
[TREXTRAS=trattributes] [TDEXTRAS=tdattributes]
[LABELPREFIX=prefix] [LABELSUFFIX=suffix] [COLUMNS=number]
[ROWS=number] [ORDER=columns|rows] [ENCODING=encoding]>
```

Description

<@CHOICELIST> allows you to easily create HTML selection list boxes, pop-up menus/drop-down lists, and radio button clusters using data from variables, database values, and so on.

This Meta Tag accepts all the attributes of the standard HTML <SELECT> tag and of the <INPUT TYPE=radio> tags. It also accepts additional attributes for specifying the values in the list and the selected item(s). Radio button groups are always formatted as a table, and an additional series of attributes defines how the radio button group table is to be formatted.

The TYPE attribute defines the type of choice list to create. This is one of SELECT or RADIO (which can be abbreviated as S and R). SELECT is the default if nothing is specified.

The following attributes of the <@CHOICELIST> tag function in the same way as the attributes of the HTML <SELECT> tag or <INPUT TYPE=radio> tag:

Attribute	Definition
NAME	The name of the <SELECT> tag or <INPUT TYPE=radio> tags, used for referencing in a <FORM> (control name).
CLASS	This attribute assigns a class name or set of class names. Any number of elements may be assigned the same class name or names. Multiple class names must be separated by white space characters.
onBlur	The onBlur event occurs when an element loses focus either by the pointing device or by tabbing navigation.
onChange	The onChange event occurs when a control loses the input focus and its value has been modified since gaining focus.

Attribute	Definition
onFocus	The onFocus event occurs when an element receives focus either by the pointing device or by tabbing navigation.
onClick	The onClick event occurs when the pointing device button is clicked over an element.
STYLE	This attribute specifies style information for the current element.

The OPTIONS attribute specifies an array of option names to appear in the selection list or radio button group. The array may have either a single column (one option name in each row) or a single row (one option name in each column).

The VALUES attribute defines an optional array of option values. If specified, the size of the array must match the one specified in the OPTIONS attribute. Each array element becomes the value for its corresponding element in the OPTIONS array. If this attribute is not specified, the value for each option is the same as its name.

The SELECTED attribute defines a single value or an array of values to be selected in the list. The value(s) must match items appearing in the VALUES attribute, if specified, or the OPTIONS attribute if VALUES is not specified. Items in this array are selected in the displayed selection list or radio button group.

The OPTIONEXTRAS attribute can be used to set additional <OPTION> tag attributes or <INPUT TYPE=radio> tag attributes. The value of this attribute is placed without parsing in the HTML <OPTION> tag or <INPUT TYPE=radio> tag. For example, OPTIONEXTRAS='CLASS="fred"' adds the CLASS="fred" attribute to each <OPTION> tag or <INPUT TYPE=radio> tag.

The following attributes apply only to lists:

- The SIZE attribute specifies the number of rows in the list that should be visible at the same time.
- The MULTIPLE attribute allows multiple selections.
- The SELECTEXTRAS attribute can be used to set additional <SELECT> tag attributes. The value of this attribute is placed without parsing in the HTML <SELECT> tag. For example, EXTRAS='ID="alpha"' adds the ID="alpha" attribute to the <SELECT> tag.

The following attributes apply only to radio button groups:

- The TABLEEXTRAS attribute sets a value that is added to the TABLE tag for the radio cluster.

- The TREXTRAS attribute sets a value that is added to each TR tag for the radio cluster.
- The TDEXTRAS attribute sets a value that is added to each TD tag for the radio cluster.
- The LABELPREFIX attribute sets a value that is prefixed to each radio button label.
- The LABELSUFFIX attribute sets a value that is appended to each radio button label.
- The COLUMNS attribute sets the number of columns of radio buttons in a radio cluster. If COLUMNS is specified, ROWS is ignored. If neither ROWS nor COLUMNS is specified, then a single-column cluster is created.
- The ROWS attribute sets the number of rows of radio buttons in a radio cluster. If COLUMNS is specified, this attribute is ignored.
- The ORDER attribute sets the direction in which the options are displayed. This attribute has two possible values: COLUMNS means each column (left to right) is filled first; ROWS means each row (top to bottom) is filled first. COLUMNS is the default value of this attribute. This attribute is used only if more than one column or row is generated.

The ENCODING attribute works slightly differently for the <@CHOICELIST> Meta Tag: the default encoding for this Meta Tag is NONE; that is, no escaping of special characters is done for the result of the Meta Tag; however, this tag does do encoding (always) as part of its normal operation; that is, any special characters within the arrays that define the options list are escaped for HTML. For example, if you specified a list of operators in the options list (= [equals];< [less than];> [greater than]), the characters that have special meaning within HTML (the less-than and greater-than characters) would be encoded as < and >, which are special HTML escape sequences. This appears correctly in a Web browser; that is, as "<" and ">".

Examples

The following TeraScript Meta Tags appear in an application file:

```
<@ASSIGN NAME=colors VALUE=<@ARRAY
VALUE="red;green;blue;yellow;black;white;">
SCOPE=request>

<@ASSIGN NAME=selectedColor VALUE="red"
SCOPE=request>

<@CHOICELIST NAME=colors SIZE=1 OPTIONS=@@colors
SELECTED=@@selectedColor>
```

On execution of the TeraScript application file, the <@CHOICELIST> Meta Tag evaluates to the following:

```
<SELECT NAME=colors SIZE=1>
<OPTION SELECTED>red
<OPTION>green
<OPTION>blue
<OPTION>yellow
<OPTION>black
<OPTION>white
</SELECT>
```

You can create a drop-down list from an existing array; for example, the resultSet of a TeraScript action:

```
<@CHOICELIST NAME=myDropDown SIZE=1
OPTIONS=@@resultSet[*,1]>
```

The following is a radio button example using the same variable arrays:

```
<@CHOICELIST NAME=colorChoice TYPE="RADIO"
VALUES=@@colors SELECTED=@@selectedColor>
```

On execution of the TeraScript application file, the <@CHOICELIST> Meta Tag evaluates to the following:

```
<TABLE><TR>
<TD><INPUT type="RADIO" name=colorChoice value="red"
CHECKED>red </TD></TR>
<TR><TD><INPUT type="RADIO" name=colorChoice
value="green">green </TD></TR>
<TR><TD><INPUT type="RADIO" name=colorChoice
value="blue">blue</TD> </TR>
<TR><TD><INPUT type="RADIO" name=colorChoice
value="yellow">yellow</TD></TR>
<TR><TD><INPUT type="RADIO" name=colorChoice
value="black">black</TD></TR>
<TR><TD><INPUT type="RADIO" name=colorChoice
value="white">white</TD></TR></TABLE>
```

The following example shows the use of table-formatting attributes and label attributes for the radio button group.

```
<@CHOICELIST NAME=colorChoice TYPE="RADIO"
VALUES=@@colors SELECTED=@@selectedColor
TABLEEXTRAS="CELLPADDING=2" labelprefix="<FONT
FACE=arial SIZE=1>" labelsuffix="</FONT>">

<TABLE CELLPADDING=2>
<TR><TD><INPUT type="RADIO" name=colorChoice
value="red" CHECKED><FONT FACE=arial SIZE=1>red</
FONT></TD> </TR>
<TR><TD><INPUT type="RADIO" name=colorChoice
```

```
value="green"><FONT FACE=arial SIZE=1>green</FONT>
</TD></TR>
<TR><TD><INPUT type="RADIO" name=colorChoice
value="blue"><FONT FACE=arial SIZE=1>blue</FONT>
</TD></TR>
<TR><TD><INPUT type="RADIO" name=colorChoice
value="yellow"><FONT FACE=arial SIZE=1>yellow</
FONT></TD></TR>
<TR><TD><INPUT type="RADIO" name=colorChoice
value="black"><FONT FACE=arial SIZE=1>black</FONT>
</TD></TR>
<TR><TD><INPUT type="RADIO" name=colorChoice
value="white"><FONT FACE=arial SIZE=1>white</FONT>
</TD></TR>
</TABLE>
```

The following example shows the use of the COLUMNS attribute for formatting the returned radio button table, returning a two-column table:

```
<@CHOICELIST NAME=colorChoice TYPE="RADIO"
VALUES=@@colors SELECTED=@@selectedColor COLUMNS=2>

<TABLE><TR><TD><INPUT type="RADIO" name=colorChoice
value="red" CHECKED>red</TD>
<TD><INPUT type="RADIO" name=colorChoice
value="yellow">yellow</TD></TR>
<TR><TD><INPUT type="RADIO" name=colorChoice
value="green">green</TD>
<TD><INPUT type="RADIO" name=colorChoice
value="black">black</TD></TR>
<TR><TD><INPUT type="RADIO" name=colorChoice
value="blue">blue</TD>
<TD><INPUT type="RADIO" name=colorChoice
value="white">white</TD></TR>
</TABLE>
```

See Also

Encoding Attribute page 8

<@CIPHER>

Syntax

```
<@CIPHER ACTION=action TYPE=type STR=string
[KEY=key][KEYTYPE]
[ENCODING=encoding]>
```

Description

Performs encryption, decryption, and hashes on strings using various algorithms and keys.

<@CIPHER> provides the TeraScript user with access to various encryption algorithms. The user may specify different keys, if required.

Three attributes are required: ACTION, TYPE, and STR.

- ACTION is the action you want to perform, for example, encrypt or decrypt.
- TYPE is the type of action you want to perform, for example, BitRoll.

Note There is a special case in which TYPE is not required. This occurs when the ACTION is Hash, and this is because TeraScript supports only one type of Hash.

- STR is the string upon which you want to execute the action, for example, a social security number. A zero length STR is processed by the underlying cipher routines.

KEY and KEYTYPE may be required or prohibited depending on the TYPE of cipher requested. Keys for some ciphers are case sensitive.

Warning messages are logged if attributes needed are missing:

```
[Warning] CIPHER: no action specified
```

```
[Warning] CIPHER: type not specified or unknown
```

```
[Warning] CIPHER: specified key not valid for this
cipher
```

The ACTION has two directions, forward and reverse. This means that you can take a string and encrypt, encode, encipher or hash it in the forward direction, and, for the reverse direction, you can decrypt, decode or decipher.

Hash is a one-way cipher: it works only in the forward position. An example use for this would be a passwords for a UNIX system. One-way hash functions are handled as encipher operations with no

corresponding decipher operation. The keyword HASH is accepted as an ACTION for this purpose.

Certain synonyms for the ciphering operations are supported:

plaintext -> ciphertext	ciphertext -> plaintext
encrypt	decrypt
encipher	decipher
encode	decode

Ciphers Supported

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

Cipher names are case insensitive. The following tables lists a short description of each cipher.

Type	Short Description
BitRoll	Swaps position of first 3 and last 5 bits in a byte
Caesar	Rotate chars by value positions mod 26
OneTimePad	Rotate characters by x positions, x being successive case-insensitive characters of key, a=1, b=2, ...
Rot13	Rotate characters by 13 positions
[MD5]	MD5 one way hash. Produces a 32 character string. ©RSA Data Security Inc. MD5 Message-Digest Algorithm.
SHA SHA256 SHA384 SHA512	Secure Hash Algorithm approved by the US Federation Information Processing Stanards (FIPS) see http://csrc.nist.gov/cryptoToolKit.
MD5MAC HMAC_SHA	Symmetric Key algorthithms used to create a Message Authentication Code when used with a specified hash algorithm.
TripleDES TripleDES3	Multiple of 8 bytes if encryption, text string if decryption. See http://csrs.nist.gov/CryptoToolKit

Type	Short Description
Rijndael	Multiple of 16 bytes if encryption, text string if decryption. Also known as AES. See http://csrs.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, text string 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	variable lenghth if encoding text string if decoding. Base 64 Encoding of Data.

Cipher Types

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

Type	Action	Key Restrictions	Key Type
BitRoll	encrypt / decrypt	prohibited	n/a
Caesar	encrypt / decrypt	optional, integer (positive and negative) values only, use "3" as default	n/a
OneTimePad	encrypt / decrypt	required, all alphabetic (no spaces or punctuation)	n/a
Rot13	encrypt / decrypt	prohibited	n/a
[MD5]	hash	ignored	n/a
SHA	hash	ignored	n/a
SHA256	hash	ignored	n/a
SHA384	hash	ignored	n/a
SHA512	hash	ignored	n/a

Type	Action	Key Restrictions	Key Type
MD5MAC	hash	required ie: KEY=00112233445566 778899aabbccddeeff" KEYTYPE="HEX" KEY=0123456789abcd ef" KEYTYPE="TEXT"	Text/Hex Default:Text
HMAC_SHA	hash	requred - variable length	Text/Hex default is Text
TripleDES	encrypt / decrypt	required 16 byte if KeyType is Text 32 byte if KeyType is Hex	Text/Hex default is Text
TripleDES3	encrypt / decrypt	required 24 byte if KeyType is Text 48 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	encode / decode	n/a	n/a
Base64	encode / decode	n/a	n/a

Security Issues

It is up to the user to guarantee the security of their information. `BitRoll`, `Caesar`, and `Rot13` are not secure at all, and `OneTimePad` is only as secure as the keys are managed and generated.

Submitting a key through a form may be insecure, especially because the HTTP request could be viewed in transit. The key and algorithm—and anything else as part of the request—can be viewed in transit. Secure channels must be used to hide text in-transit, and very strong ciphers must be used to guarantee security.

Examples

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

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

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

See Also

`Encoding Attribute` page 8

<@CLASSFILE>

Syntax

```
<@CLASSFILE [ENCODING=encoding]>
```

Description

Returns the path to the current TeraScript class file, including the file name. This Meta Tag is useful for debugging. The path returned is relative to the Web server root directory.

Outside of a method call, this Meta Tag returns nothing.

See Also

Encoding Attribute	page 8
<@APPFILEPATH>	page 24
<@CLASSFILEPATH>	page 72
TCFSearchPath	page 464

<@CLASSFILEPATH>

Syntax

```
<@CLASSFILEPATH [ENCODING=encoding]>
```

Description

Returns the path to the current TeraScript class file, excluding the TeraScript class file name, but including the trailing slash.

This Meta Tag is useful for creating links that reference an application file, <@INCLUDE> file, or image file in the same directory as the currently executing TeraScript class file.

The path returned is relative to the Web server root directory.

Outside of a method call, this Meta Tag returns nothing.

Examples

```
<A
HREF="<@CGI><@CLASSFILEPATH>homer.taf?function=form
">
</A>
```

This example calls the `homer.taf` file, located in the same directory as the currently executing TeraScript class file.

```
<@INCLUDE FILE="<@CLASSFILEPATH>header.html">
```

This example includes the `header.html` file, located in the same directory as the currently-executing TeraScript class file.

```
<IMG SRC="<@CLASSFILEPATH>logo.gif">
```

This example references the `logo.gif` file, located in the same directory as the currently executing TeraScript class file.

See Also

Encoding Attribute	page 8
<@APPFILE>	page 22
<@CGI>	page 58
<@CLASSFILE>	page 71
<@INCLUDE>	page 170
TCFSearchPath	page 464
<@URLENCODE>	page 287

<@CLEARERRORS>

Description

This Meta Tag may be used in an action's Error HTML or the `error.htx` file. It removes all accumulated errors and allows TeraScript Server to resume processing, starting with the next action. When called from anywhere but Error HTML or the `error.htx` file, this Meta Tag is ignored.

This Meta Tag has no attributes and returns no value.

When <@CLEARERRORS> is called the http status code and reason phrase are reset back to 200 OK.

Example

<@CLEARERRORS> clears the automatically-generated TeraScript error and allows you to continue processing.

As well, you could insert your own error text to be returned to the user.

See Also

`defaultErrorFile`	page 408
<@ERRORS> </@ERRORS>	page 145
<@HTTPSTATUSCODE>	page 160
<@HTTPREASONPHRASE>	page 159
<@THROWERROR>	page 263

<@COL>

Syntax

<@COL [NUM=*number*] [ENCODING=*encoding*] [FORMAT=*format*]>

Description

Returns the value of the column NUM in the current record of a result rowset or array. This tag may be used in any Results HTML. <@COL NUM=1> refers to the first column in the current row, <@COL NUM=2> the second, and so on.

This tag is generally used in a <@ROWS> block. Outside of a <@ROWS> block, this tag behaves like <@COLUMN> that is, it returns the value of the column NUM for the *first* row of the current result rowset or array. This Meta Tag can be used with no attributes inside a <@COLS> block. In this case, it returns the value of the current column.

Example

```
<@ROWS>
Column 1:<@COL NUM=1><BR>
Column 2:<@COL NUM=2><BR>
Column 3:<@COL NUM=3><BR>
</@ROWS>
```

This prints the values from columns one, two and three for each row in the current rowset.

See Also

<@COLS> </@COLS> page 75
<@COLUMN> page 76
Encoding Attribute page 8
Format Attribute page 11
<@ROWS> </@ROWS> page 239

<@COLS> </@COLS>

Syntax

```
<@COLS></@COLS>
```

Description

Processes the enclosed HTML once for each column in the current row.

Text appearing between <@COLS></@COLS> is processed once for each column in the current row of a <@ROWS> block. If a <@ROWS> block appears between these tags, <@ROWS> is ignored.

This tag block is very useful for looping through an unknown number of columns, such as might be generated by a Direct DBMS action with variable SQL.

Example

```
<@ROWS>
 <@COLS>
    <@COL>
 </@COLS>
 <BR>
</@ROWS>
```

This example would return every column in every row returned by the Search action that it is attached to.

See Also

<@COL>	page 74
<@CURCOL>	page 86
<@NUMCOLS>	page 213
<@ROWS> </@ROWS>	page 239

<@COLUMN>

Syntax

<@COLUMN NAME=*name* [ENCODING=*encoding*] [FORMAT=*format*]>

Description

Returns the value of the named column in the current row of a <@ROWS> block. The name can be in `column`, `table.column` or `owner.table.column` format, as long as it is not ambiguous. This Meta Tag is only valid for Search and Direct DBMS actions.

Outside of a <@ROWS> block, this Meta Tag returns the value of the named column for the first row of the current result set.

If the tag cannot be evaluated due to insufficient information (ambiguity) or a mismatch for all the columns, a blank is returned.

This tag is supported for Direct DBMS actions only when ODBC data sources are used.

Example

```
<@ROWS>
        <@COLUMN NAME=TEST.TEST_TABLE_A.KEY_FIELD>,
        <@COLUMN NAME=TEST.TEST_TABLE_A.CHAR_FIELD>,
        <@COLUMN NAME=INT_FIELD>
        <BR>
</@ROWS>
```

This example goes through every row in the results set and returns the values of the named columns in each row.

See Also

<@COL> page 74
<@COLS> </@COLS> page 75
<@CURCOL> page 86
Encoding Attribute page 8
Format Attribute page 11
<@ROWS> </@ROWS> page 239

<@COMMENT> </@COMMENT>

Syntax

```
<@COMMENT>comment</@COMMENT>
```

Description

This tag block gives you the ability to place large comments in files (taf, tcf, tml, include) processed by TeraScript.

It is intended as a means of notation for multiple programmers who may access the same application files, or as a notation for a single user managing large application files and projects. It is valid in Results, No Results, and Error HTML, and in Direct DBMS, SQL and Script action scripts.

The material inside these tags is stripped out and never appears in HTML sent to the user's Web browser.

Note These tags are required to appear in pairs, and unpaired appearances are treated as unrecognized tags and left untouched.

Examples

```
<@COMMENT> This function does this </@COMMENT>
```

The tag and the HTML contained inside are removed before the rest of the HTML is returned to the user.

```
<@COMMENT> do this: <@ASSIGN NAME=myVar
VALUE="asdfasd"> </@COMMENT>
```

The tag and the HTML contained inside are removed before the HTML is returned, and <@ASSIGN> is not an executed part of the application file.

See Also

<@!> page 309

<@CONFIGPATH>

Description

This Meta Tag returns the full path to the configuration directory in use by TeraScript Server. This directory is by default where the TeraScript Server configuration files are located, such as `terascript.ini,clients.ini, handlers.ini` etc.

The configuration directory varies according to the platform. For more information, see `A Note on Default Locations <page $pagenum>`.

The path returned includes the trailing directory separator.

Security Feature

If a user scope `configPasswd` variable with the same value as the system `configPasswd` does not exist, an error is generated.

Example

If `<@CONFIGPATH>` is used in an application file on a machine where TeraScript Server is installed into the default directory, it returns:

On Windows:

```
C:\Program Files\TeraScript\Server\Configuration
```

On Linux:

```
usr/local/TeraScript/configuration
```

On OS X:

```
/Applications/TeraScript/Server/configuration
```

<@CONNECTIONS>

Syntax

<@CONNECTIONS [DSN=*datasourcename*] [TYPE=*ODBC*|*DAM*|*FileMaker*|*Oracle*|*DLL*|*CommandLine*|*Java*|*AppleEvent*|*Mail*] [ENCODING=*encoding*] [{*array attributes*}]>

Description

The <@CONNECTIONS> Meta Tag provides information about each data source object currently in use by TeraScript Server. TeraScript Server considers External action connections and Mail action connections to be data sources, so information on these actions is also returned.

<@CONNECTIONS> returns a two-dimensional array with one row for each data source object (the optional attributes may be used to restrict the data source, External action, or Mail action information returned by the Meta Tag). Usually, data source objects represent connections. However, in some situations, TeraScript Server shares a single database connection between multiple data source objects. In this case, the returned NumConnections value displays the total number of shared connections.

The optional DSN attribute is used to restrict the information returned to that from data sources, External actions, or Mail actions with the specified name. If omitted, information for all data sources, External actions, or Mail actions is returned. If an invalid name is specified, an error is returned.

The optional TYPE attribute is used to restrict the information returned to data sources with the specified type, External actions of the specified type, or Mail actions. The type must be one of the listed values. If omitted, information for all data source types, External actions, and Mail actions is returned. If an invalid type is specified, an error is returned.

There are array-returning attributes that can be used to specify prefixes and suffixes for the returned array, rows within the array, and columns within the rows. They are described in the section Array-to-Text Attributes <page $pagenum>. By default, the returned array is formatted as an HTML table.

Mail and External actions are treated as data sources by TeraScript Server. Specifying DLL (DLL-type External action), CommandLine (command-line External action), Java (Java External action), or Mail (Mail action) in the TYPE attribute returns information on these actions.

<@CONNECTIONS> returns a two-dimensional array, containing the following columns:

Category	Description
Version	Version of <@CONNECTIONS> information. Will return "1" for the initial version of this Meta Tag.
Name	Data source: returns the name of the data source. Mail actions: returns the host/port of the mail server. Command line: returns the name of the last command to be executed. DLL: returns the name of the DLL.
Type	Data source, External action, or Mail action type (ODBC | JDBC | FileMaker | Oracle | DLL | CommandLine | Java | Mail).
Info	Additional information about the connection (for example, ODBC driver name and version). The information returned depends on the data source driver. Empty for External and Mail actions.
UserName	User name (after tag substitution) used to connect to the data source. Blank if no user name was specified.
NumConnections	Number of physical connections to this data source. Not used for External and Mail actions.
MinutesOpened	Number of minutes the connection to the data source has been opened.
MinutesIdle	Number of minutes since a query was last executed by the connection.
LastUsedBy	The user reference of the last user whose query was processed by this connection.
ErrorsGenerated	Number of errors generated during execution.

Note Row 0 of the array returned by <@CONNECTIONS> contains the column titles listed in the Category column of the above table. You can return the category names by using the following Meta Tags:

```
<@ASSIGN request$tempArray value=<@CONNECTIONS>>
<@VAR request$tempArray[0,*]>
<@VAR request$tempArray>
```

Example

```
<@CONNECTIONS>
```

Returns a two-dimensional array listing all open connections for all data sources.

```
<@CONNECTIONS TYPE=ODBC>
```

Returns a two-column array listing all open ODBC data sources.

See Also

Array-to-Text Attributes page 17
Encoding Attribute page 8

<@CONTINUE>

Description

Terminates execution of the current iteration of a <@COLS>, <@ROWS>, <@FOR>, or <@OBJECTS> block. Execution of the loop continues from the beginning of the block. Outside of a <@COLS>, <@ROWS>, or <@FOR> block, this tag does nothing. <@CONTINUE> has no attributes.

This tag is generally used with an <@IF> tag to terminate the current iteration of a loop when some condition is met. Be careful to handle nested loops properly: only the innermost loop's processing is affected by the continue command.

Example

```
Only public records will be shown.
<@ROWS>
<HR>
Here are the values from record <@CURROW> of the
results:<P>
<@IF EXPR="<@COL TYPE>='internal'"
TRUE="<@CONTINUE>">
<STRONG>Name:</STRONG> <@COLUMN
NAME="contact.name"><BR>
<STRONG>Phone:</STRONG> <@COLUMN
NAME="contact.phone"><BR>
</@ROWS>
End of records.
```

This example suppresses the printing of the records where the type column has the value "internal". If the type column has the value "internal", the loop processing goes directly to the </@ROWS> tag (and then to the beginning of the loop if there are more records).

See Also

<@BREAK> page 42
<@COLS> </@COLS> page 75
<@EXIT> page 147
<@FOR> </@FOR> page 151
<@OBJECTS></@OBJECTS> page 217
<@ROWS> </@ROWS> page 239

<@CREATEOBJECT>

Syntax

```
<@CREATEOBJECT TYPE=type OBJECTID=objectID [EXPIRYURL=url]
[INITSTRING=string] [SYSTEMOBJECT=true|false]>
```

Description

This Meta Tag creates a new instance of a particular object. It must be used in conjunction with the <@ASSIGN> Meta Tag or the Assign action.

The TYPE attribute defines the name of any valid TeraScript object handler: COM, JavaBean, or TCF (TeraScript class file).

The OBJECTID attribute defines the object: for COM, the ProgID or ClassID; for TeraScript class files, the file name; for JavaBeans, the name of the JavaBean.

Note The object you specify in the OBJECTID attribute must be able to be located by TeraScript Server; that is, for TeraScript class files, the TCFSearchPath configuration variable must specify the path to the named TeraScript class file; for a JavaBean, the CLASSPATH environment variable must contain the path to the JavaBean. For COM objects, the object must be registered on the machine where TeraScript Server is running.

The object must also not be disallowed from running in the object configuration file.

The EXPIRYURL attribute defines a URL to call when the variable expires.

Meta Tags are allowed in all attributes. This tag does not return anything.

COM-specific Attributes

The INITSTRING attribute is used for COM only, and defines the object-specific string to use for initialization. The SYSTEMOBJECT attribute is used for COM only, and can have the value TRUE or (the default) FALSE. If true, TeraScript gets an existing instance from the system rather than creating a new one.

Example

```
<@ASSIGN NAME=myObject SCOPE=request
VALUE=<@CREATEOBJECT TYPE=COM
OBJECTID=WitangoOM.clsWitangoOM>>
```

In this example the Meta Tags create an object instance variable called `myObject` in request scope and creates an object instance of the COM object `WitangoOM.clsWitangoOM`.

See Also

`<@CALLMETHOD>`	page 55
`<@GETPARAM>`	page 154
`<@NUMOBJECTS>`	page 214
`<@OBJECTAT>`	page 216
`<@OBJECTS></@OBJECTS>`	page 217

<@CRLF>

Description

Evaluates to a carriage return/linefeed combination as required by the HTTP RFC (RFC-2626).

This tag is used in the HTTP header specified by the `headerFile` configuration variable to generate the line terminators required for the HTTP header.

The external HTTP header file (by default ***header.htx***) should use <@CRLF> and should NOT contain any OS line breaks.

Example

```
<@ASSIGN name="httpHeader" scope="request" value="HTTP/
1.1 301 Moved Permanently<@CRLF>content-type: text/
html<@CRLF><@CRLF>">
```

See Also

`headerFile`	page 424
`httpHeader`	page 425
`<@LITERAL>`	page 197

<@CURCOL>

Description

Returns the index (1, 2, 3, ...) of the column currently being processed if placed inside a <@COLS></@COLS> block.

Example

```
<@ROWS>
  <@COLS>
   <@CURCOL>
  </@COLS>
 <BR>
</ROWS>
```

If this example looped through two three-column rows, it would return:

```
1 2 3
1 2 3
```

See Also

<@COL>	page 74
<@COLS> </@COLS>	page 75
<@NUMCOLS>	page 213
<@ROWS> </@ROWS>	page 239

<@CURRENTACTION>

Syntax

<@CURRENTACTION [ENCODING=*encoding*]>

Description

Returns the name of the currently executing action. This Meta Tag can be useful for debugging application files.

Example

```
<@ASSIGN NAME=<@CURRENTACTION>_RowCount
VALUE=<@NUMROWS>>
```

This text could be saved in a text file and included with <@INCLUDE> to assign the number of rows returned by the action to a variable whose name includes the action name.

See Also

Encoding Attribute page 8

<@CURRENTDATE>, <@CURRENTTIME>, <@CURRENTTIMESTAMP>

Syntax

```
<@CURRENTDATE [ENCODING=encoding] [FORMAT=format]>
<@CURRENTTIME [ENCODING=encoding] [FORMAT=format]>
<@CURRENTTIMESTAMP [ENCODING=encoding] [FORMAT=format]>
```

Description

Returns the current date, time, or timestamp (date and time concatenated). If FORMAT is specified, it is used to format the value; otherwise, the default date and time formats specified by the date and time configuration variables are used.

For more information, see "Date and Time Formatting Codes" on page 402.

Codes for the elements of FORMAT are shown in the description of the date and time formatting configuration variables. Date and time values returned by these Meta Tags reflect the setting of the clock on the computer where TeraScript Server is installed.

Examples

```
Today is <@CURRENTDATE>
```

This prints a message that includes the current date in the format specified by the default date format.

```
It is now <@CURRENTTIME FORMAT="datetime:%H:%M:%S">
```

This prints a message that includes the current time in 24-hour format.

```
It is day <@CURRENTDATE FORMAT="%j"> of <@CURRENTDATE
FORMAT="%Y">
```

This prints a message that includes the current day and year.

See Also

dateFormat	page 401
Encoding Attribute	page 8
<@FORMAT>	page 153
Format Attribute	page 11
timeFormat	page 401
timestampFormat	page 401

<@CURROW>

Description

Returns the number of the current row being processed in a <@ROWS> or <@FOR> block. It evaluates to "0" before or after a <@ROWS> block.

Example

```
<@ROWS>
<HR>
Here are the values from record <@CURROW> of the
results:<P>
<STRONG>Name:</STRONG> <@COLUMN
NAME="contact.name"><BR>
<STRONG>Phone:</STRONG> <@COLUMN
NAME="contact.phone"><BR>
</@ROWS>
```

Prior to displaying each contact's name and phone number, the number of the record in the current rowset is displayed.

See Also

<@ABSROW> page 18
<@FOR> </@FOR> page 151
<@NUMROWS> page 215
<@ROWS> </@ROWS> page 239

<@CUSTOMTAGS>

Syntax

<@CUSTOMTAGS [SCOPE=*system*|*application*] [{*array attributes*}]>

Description

This Meta Tag returns a three-column array of all custom Meta Tags in the scope specified. For more information on custom Meta Tags, see Custom Meta Tags <page $pagenum>.

If no scope is specified, all custom Meta Tags are returned. The columns of the array are Tag Name, Package Name, and Scope. These names are put in row 0 of the array. No password is required to use this tag.

Example

If a custom tag package called Small Demo App is installed, creating a custom Meta Tag <@COMTESTOBJECT>, available in system scope, <@CUSTOMTAGS> returns:

COMTESTOBJECT	Small Demo App	System

See Also

Array-to-Text Attributes page 17
<@RELOADCUSTOMTAGS> page 234
Custom Meta Tags page 305

<@DATASOURCESTATUS>

Syntax

<@DATASOURCESTATUS [DSN=*datasourcename*] [TYPE=*ODBC*|*DAM*|*FileMaker*|*Oracle*|*DLL*|*CommandLine*|*Java*|*AppleEvent*|*Mail*] [ENCODING=*encoding*] [{*array attributes*}]>

Description

The <@DATASOURCESTATUS> Meta Tag returns a two-dimensional array containing summary information about data sources used by TeraScript Server. The Meta Tag returns one row for each data source currently in use and for data sources used previously but whose connections have expired or have been closed. TeraScript Server considers External action connections and Mail action connections to be data sources, so information on these actions is also returned.

The optional DSN attribute is used to restrict the information returned to that from data sources, External actions, or Mail actions with the specified name. If omitted, information for all data sources, External actions, or Mail actions is returned. If an invalid name is specified, an error is returned.

The optional TYPE attribute is used to restrict the information returned to data sources, External actions, or Mail actions of the specified type. The type must be one of the listed values (ODBC, JDBC, FileMaker, or Oracle, if data sources; DLL, CommandLine, Java, or Mail, if referring to actions). If omitted, information for all data source types, External action types, and Mail actions is returned. If an invalid type is specified, an error is returned.

There are array-returning attributes that can be used to specify prefixes and suffixes for the returned array, rows within the array, and columns within the rows. They are described in the section Array-to-Text Attributes <page $pagenum>. By default, the returned array is formatted as an HTML table.

Mail and External actions are treated as data sources by TeraScript Server. Specifying DLL (DLL-type External action), CommandLine (command-line External action), Java (Java External action), or Mail (Mail action) in the TYPE attribute returns information on these actions.

(The <@CONNECTIONS> Meta Tag differs from the <@DATASOURCESTATUS> Meta Tag in that it returns one row

containing information about each data source connection currently in use by TeraScript Server.)

Note This tag is limited to returning information about the data source types listed above. Information about other data sources used by TeraScript Server (such as those accessed through external actions) are not returned by this Meta Tag.

The <@DATASOURCESTATUS> Meta Tag returns a two-dimensional array containing the following columns:

Category	Description
Version	Version of <@DATASOURCESTATUS> information. Returns "1" for the initial version of this Meta Tag.
Name	For database data sources, the name of the data source; for others, the name of the resource connecting to (for example, Mail server or External action.
Type	Data source type (ODBC | FileMaker | Oracle | JDBC | CommandLine | Java | Mail).
Info	Additional information about the connection (for example, ODBC driver name and version). The information returned depends on the data source driver. Empty for External and Mail actions.
ThreadSafe	Returns "1" if driver is single-threaded (as specified using the `dsconfig` configuration variable) or "0" if driver is thread safe.
UserName	User name (after tag substitution) used to connect to the data source. Blank if no user name was specified.
MaxConnections	Maximum number of connections allowed for this data source, as specified using the `dsconfig` configuration variable. If `dsconfig` is not being used to limit this data source, then "0" (unlimited) is returned.
NumConnections	Number of connections currently open to this data source - this will always be one as only once active ODBC connection is allowed per datasource.
NumExpiredConnections	Number of connections previously opened to this data source, but closed or expired.
MinutesIdle	Number of minutes since a query was last executed by the connection.

Category	Description
NumQryExecuted	Number of queries executed by the data source.
MaxQryProcessTime	Maximum time (in milliseconds, accurate to 1/60th of a second) required to process a query. Does not include time to fetch or process results.
AvgQryProcessTime	Average time (in milliseconds, accurate to 1/60th of a second) required to process a query. Does not include time to fetch or process results.
MaxRowSetSize	Maximum number of rows returned by a single query processed by the connection.
AvgRowSetSize	Average number of rows returned by a single query processed by the connection.
ErrorsGenerated	Number of errors generated during execution.

Note Row 0 of the array returned by <@DATASOURCESTATUS> contains the column titles listed in the Category column of the above table. You can return the category names by using the following Meta Tags:

```
<@ASSIGN request$tempArray
value=<@DATASOURCESTATUS>>
<@VAR request$tempArray[0,*]>
<@VAR request$tempArray>
```

Example

```
<@DATASOURCESTATUS>
```

Returns a two-dimensional array listing all data source categories for all data sources.

```
<@DATASOURCESTATUS TYPE=ODBC>
```

Returns a two-column array listing information for only ODBC data sources.

See Also

Array-to-Text Attributes page 17
<@CONNECTIONS> page 79
Encoding Attribute page 8

<@DATEDIFF>

Syntax

<@DATEDIFF DATE1=*firstdate* DATE2=*seconddate* [FORMAT=*format*]>

Description

Returns the number of days between the two dates specified.

<@DATEDIFF> handles ODBC, ISO, some numeric formats, and textual formats.

If the date is entered incorrectly—wrong separators or wrong values for year, month or day—the tag returns "Invalid date!".

The date attributes are mandatory. If no attribute is found while the expression is parsed, the tag returns "No attribute!".

All formats assume the Gregorian calendar. All years must be greater than zero.

For more information, see "<@ISDATE>, <@ISTIME>, <@ISTIMESTAMP>" on page 176.

Note When a two-digit year is given, the following centuries are assumed:

Value	Century
00-36	2000s
37-99	1900s

For example, a two-digit year of 99 is evaluated as 1999, and a two-digit year of 00 is evaluated as 2000.

Example

```
<@DATEDIFF DATE1=1998-02-20 DATE2=1998-02-27>
```

The date returned is calculated as DATE1 minus DATE2. This example tag returns "-7", the number of days between the two dates.

See Also

<@DAYS> page 97
<@FORMAT> page 153
Format Attribute page 11
<@ISDATE> page 176
<@ISTIME> page 176
<@ISTIMESTAMP> page 176

<@DATETOSECS>, <@SECSTODATE>

Syntax

<@DATETOSECS DATE=*date* [FORMAT=*format*]>

<@SECSTODATE SECS=*seconds* [ENCODING=*encoding*] [FORMAT=*format*]>

Description

<@DATETOSECS> checks the entered date and, if valid, converts it into seconds using as a reference—midnight (00:00:00) January 1, 1970 (1970-01-01).

Conversely, <@SECSTODATE> checks the entered seconds and converts them to a date.

These tags support dates in the range 1970–2037.

Both tags handle ODBC, ISO, and some numeric formats.

If the date is entered incorrectly—wrong separators or wrong values for year, month, or day—the tag returns "Invalid date!".

The date attribute is mandatory. If no attribute is found while the expression is parsed, the tag returns "No attribute!".

Note When a two-digit year is given, the following centuries are assumed:

Value	Century
00-36	2000s
37-99	1900s

For example, a two-digit year of 99 is evaluated as 1999, and a two-digit year of 00 is evaluated as 2000.

Examples

```
<@DATETOSECS DATE=1970-01-01>
```

This tag returns "0", the number of seconds since January 1, 1970.

```
<@DATETOSECS DATE=2000-01-01>
```

This tag returns "946684800", the number of seconds since January 1, 1970.

```
<@SECSTODATE SECS=946684800>
```

This tag returns "2000-01-01", the date derived from the number of seconds. The example assumes a `dateFormat` of "`%Y-%m-%d`".

See Also

`dateFormat`	page 401
`Encoding Attribute`	page 8
`<@FORMAT>`	page 153
`Format Attribute`	page 11
`<@ISDATE>`	page 176
`<@ISTIME>`	page 176
`<@ISTIMESTAMP>`	page 176
`<@SECSTOTIME>`	page 266
`<@SECSTOTS>`	page 275
`timeFormat`	page 401
`timestampFormat`	page 401
`<@TIMETOSECS>`	page 266
`<@TSTOSECS>`	page 275

<@DAYS>

Syntax

<@DAYS DATE=*date* DAYS=*days* [ENCODING=*encoding*] [FORMAT=*format*]>

Description

Adds the days in the DAYS attribute to the date in the DATE attribute. Use a negative DAYS value to subtract days.

All formats assume the Gregorian calendar. All years must be greater than zero.

<@DAYS> handles ODBC, ISO, and some numeric formats.

If the date is entered incorrectly—wrong separators or wrong values for year, month, or day—the tag returns "Invalid date!".

The attributes, DATE and DAY are mandatory. If no attribute is found for ether the tag returns "No attribute!".

Note When a two-digit year is given, the following centuries are assumed:

Value	Century
00-36	2000s
37-99	1900s

For example, a two-digit year of 99 is evaluated as 1999, and a two-digit year of 00 is evaluated as 2000.

Example

```
<@DAYS DATE=1998-02-20 DAYS=7>
```

This tag returns "1998-02-27", the new date, assuming the dateFormat is "%Y-%m-%d".

See Also

dateFormat page 401
<@DATEDIFF> page 94
Encoding Attribute page 8
<@FORMAT> page 153
Format Attribute page 11

<@DBMS>

Syntax

```
<@DBMS [ENCODING=encoding]>
```

Description

Returns the concatenated name and version of the database used by the current action's data source.

If the current action has no data source, the Meta Tag returns the information for the most recent data source used during the current execution of the application file. If used prior to the execution of a database-related action, this tag returns an empty string.

This tag is useful in Direct DBMS actions where you may want to execute different SQL depending on which DBMS is in use.

The exact values returned by this Meta Tag depend on values returned by the current database driver and/or server software.

Example

```
<@IFEQUAL VALUE1="<@DBMS> VALUE2="ORACLE*:>
SQL to execute only if we are connected to an Oracle
data source.
</@IF>
```

This example from a Direct DMBS action is used to specify the SQL to execute when an Oracle data source is assigned to the action.

See Also

<@DSTYPE> page 124
Encoding Attribute page 8

<@DEBUG> </@DEBUG>

Syntax

```
<@DEBUG></@DEBUG>
```

Description

These paired tags provide the TeraScript user more power to debug application files. If debugging is on, TeraScript processes the text inside the <@DEBUG></@DEBUG> pair; otherwise, these tags and the content inside are ignored.

This tag is valid in Results, No Results, and Error HTML only.

Examples

```
<@DEBUG> <@COLUMN NAME="contacts.lastname">
</@DEBUG>
```

This example includes the value of the `lastname` column of the contacts table in the HTML only if in debug mode.

```
<@DEBUG> <@ASSIGN NAME="gname"
VALUE="<@COLUMN NAME='contacts.lastname'>">
</@DEBUG>
```

This example executes the variable assignment only if in debug mode.

<@DEFINE>

Syntax

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

Description

This tag creates an empty variable of the specified type in the specified scope.

Type Attribute

The available variable types are listed below:

Text

Which is used to define a text string.

To define a text variable called *FirstName* in user scope:

```
<@DEFINE NAME="FirstName" SCOPE=”user”  TYPE=”text”>
```

Arrays

Which is used to define an array. Two optional attributes, ROWS and COLS, are available to create an array of a required size. They are ignored for all types except ARRAY.

To define an array of 1 row and five columns called *FirstNameArray* in user scope:

```
<@DEFINE NAME="FirstNameArray" SCOPE=”user”
TYPE=”array” ROWS=”1” COLS=”5”>
```

DOM (XML document)

Which is used to define a document instance (XML) variable.

OBJECT

Which is used to define a variable of an object instance.

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

Objects created with <@DEFINE> are NULL objects (<@ISNULLOBJECT> would return 1) until they are assigned a valid object.

For more information on “Working with Variables see Working With Variables <page $pagenum>.

Scope Attribute

Scoping is the method by which variables can be organized and disposed of in an orderly and convenient fashion. There are various levels of scoping, each of which has an appropriate purpose:

For more information, see "Configuration Variables" on page 373.

- **System Scope** contains any variables that are general to all users. This scope contains only TeraScript Server configuration variables. To use this scope, specify `SCOPE=system`.

For more information, see "domainScopeKey" on page 412 .

- **Domain Scope** contains variables that users can share if they are accessing a particular TeraScript application file from a specified TeraScript domain. TeraScript domains are specified in a domain configuration file, or default to the domain name (base URL or IP address) of the path to the TeraScript application file. This scope is defined by setting the system configuration variable `domainScopeKey` appropriately; that is, setting it to a value that can differentiate such users. By default, this is `<@DOMAIN>`, which returns the value of the current TeraScript domain.To use this scope, specify `SCOPE=domain`.
- **Application Scope** contains variables that are shared across TeraScript applications. TeraScript applications are defined by TeraScript users in an application configuration file. To use this scope, specify `SCOPE=application`.
- **User Scope** contains variables that a user defines and expects to be able to access from many application files or invocations of single application files. To use this scope, specify `SCOPE=user`.
- **Request Scope** contains variables that should be unique to every invocation of any application file. For example, this scope could be used for temporary variables that reformat output from a search action. All variables of this scope are removed when the application file concludes execution. To use this scope, specify `SCOPE=request`.
- **Instance Scope** contains variables that are valid in an instance of a TeraScript class file. These variables can be shared across methods called on a TeraScript class file, if the methods are called on the same instance. To use this scope, specify `SCOPE=instance`.
- **Method Scope** contains variables that should be unique to a method of a TeraScript class file. To use this scope, specify `SCOPE=method`.
- **Cookie Scope** contains variables that are sent to the user's Web browser as cookies (that is, a small text file kept by the Web

browser for a specified amount of time). To use this scope specify `SCOPE=cookie`.

- **Custom Scope** is user-specified. It is outside of the scope search hierarchy.

For more information on "Scoping" see `Understanding Scope`page 321.

If this attribute is omitted, the new variable is created in the default scope. The default scope is normally REQUEST, but can be changed by setting the `defaultScope` configuration variable in the `terascript.ini` file.

See Also

`<@ASSIGN>`	page 33
`<@VAR>`	page 292
`Working With Variables`	page 319
`variableTimeout`	page 478

<@DELROWS>

Syntax

```
<@DELROWS ARRAY=arrayVarName [POSITION=startWhere]
[NUM=numToDelete] [SCOPE=scope]>
```

Description

Deletes rows from the array in the variable named by ARRAY. This tag does not return anything. With no additional attributes specified, this tag deletes one row from the end of the array.

The POSITION attribute specifies the index of the row to start deleting from. If the value specified in POSITION is 0 or greater than the number or rows in the array, no rows are deleted. If POSITION is -1 (the default), the last row in the array is deleted.

The NUM attribute specifies the number of rows to delete. The default is 1. If this attribute specifies a range that, in combination with POSITION, exceeds the bounds of the array, only those rows that do exist in the range are deleted, and no error is returned.

The SCOPE attribute specifies the scope of the variable specified as the value of the ARRAY attribute. If the scope is not specified, the default scoping rules are used.

Meta Tags are permitted in any of the attributes.

Examples

- The request variable `colors` contains the following array:

orange
amber
red
burnt umber

```
<@DELROWS ARRAY="colors" POSITION=2 NUM=2
SCOPE="request">
```

The request variable `colors` now contains the following array:

orange
burnt umber

- The user variable `choices_list` contains the following array:

News	2
Sports	3

Movies	4
Stocks	1

```
<@DELROWS ARRAY="choices_list" SCOPE="user">
```

The user variable `choices_list` now contains:

News	2
Sports	3
Movies	4

See Also <@ADDROWS> page 20

<@DISTINCT>

Syntax

```
<@DISTINCT ARRAY=arrayVarName
[COLS=compCol [compType] [, ...]] [SCOPE=scope]
[{array attributes}]>
```

Description

Returns an array containing the distinct, or unique, rows in the input array. The original array is not altered.

The ARRAY attribute specifies the name of a variable containing an array. The COLS attribute specifies the column(s) to consider when checking for duplicate rows. Columns can be specified using either column numbers or names, with an optional comparison type specifier (*compType*).

Valid comparison types are SMART (the default), DICT, ALPHA, and NUM. DICT compares values alphabetically without considering case. ALPHA is a case-sensitive comparison. NUM compares values numerically. SMART checks whether values are numeric or alphabetic and performs a NUM or DICT comparison.

If the COLS attribute is omitted, all columns are considered using the SMART comparison type when eliminating duplicates.

Multiple columns may be specified, separated by commas. Each column specification may include a comparison type specifier. If the comparison type specification is used, it must follow the name or number of the column to be sorted, separated by a space. For example, COLS="1 NUM, 2 DICT" specifies that the first column's values are compared numerically, and the second column's values are compared alphabetically, not case-sensitive.

The SCOPE attribute specifies the scope of the variable specified as the value of the ARRAY attribute. If the scope is not specified, the default scoping rules are used.

There are array-returning attributes that can be used to specify prefixes and suffixes for the returned array, rows within the array, and columns within the rows. They are described in the section Array-to-Text Attributes <page $pagenum>. By default, the returned array is formatted as an HTML table.

Meta Tags are permitted in any of the attributes.

Examples

If the request variable `test` contains the following array:

1	a
1	a
2	a
3	b
3	b
4	c
4	c
6	d
7	e
7.0	f

`<@DISTINCT ARRAY="test" SCOPE="request">` returns:

1	a
2	a
3	b
4	c
6	d
7	e
7.0	f

`<@DISTINCT ARRAY="test" COLS="1 NUM" SCOPE="request">` returns:

1	a
2	a
3	b
4	c
6	d
7	e

`<@DISTINCT ARRAY="test" COLS="2" SCOPE="request">` returns:

1	a
3	b
4	c
6	d
7	e
7.0	f

See Also

<@FILTER>	page 148
<@INTERSECT>	page 171
<@SORT>	page 256
<@UNION>	page 277

<@DOCS>

Syntax

```
<@DOCS [FILE=appfile] [ENCODING=encoding]>
```

Description

This Meta Tag has been deprecated as of the release of TeraScript 6. It is currently operational but will be removed in the next major version release. Developers are encouraged to discontinue use of this Meta Tag. When applicable, a warning will be reported to the `witangoevents.log` file.

Displays the content of an application file or class file in HTML.

Evaluates to an action list for the named application file. Each action entry in the list is a link to a detailed description of that action. The path to the named application file must be relative to the Web server document root.

If no `FILE` attribute is provided, `<@DOCS>` evaluates to the running application file.

For more information, see `docsSwitch` `<page $pagenum>`.

There is a special configuration variable—`docsSwitch`—that can be set to on or off. It must be set to "on" for this tag to work.

This Meta Tag returns an empty value if the `FILE` attribute specifies an application file saved as run-only.

When used in Results HTML, the `ENCODING=NONE` attribute must be used in order for it to be displayed properly in the Web browser.

Examples

```
<@DOCS ENCODING=NONE>
```

```
<@DOCS FILE="/Oracle/Car_demo/car_search.taf"
ENCODING=NONE>
```

See Also

`Encoding Attribute` page 8

<@DOM>

Syntax

<@DOM VALUE=*value*>

Description

This tag is used to parse XML into a document instance. It supports DOM2, XPATH and XPOINTER.

This Meta Tag is usually used in conjunction with <@ASSIGN> or the Assign action to create a document instance variable.

The VALUE attribute specifies the XML that is to be parsed into a document instance.

Example

```
<@ASSIGN NAME="myDom" SCOPE="application"
VALUE=<@DOM VALUE="<XML><DIV><P>Paragraph 1
</P><P>Paragraph 2</P></DIV></XML>">>
```

This assigns the XML specified by <@DOM> to a document instance variable in application scope called myDom.

See Also

<@DOMDELETE>	page 111
<@DOMINSERT>	page 114
<@DOMREPLACE>	page 116
<@DOMSEARCH>	page 118
<@ELEMENTATTRIBUTE>	page 126
<@ELEMENTATTRIBUTES>	page 129
<@ELEMENTNAME>	page 131
<@ELEMENTVALUE>	page 134

<@DOMAIN>

Description

This tag returns the current domain.

TeraScript domains are configured with the domain configuration file, which can be edited through the Administration Application `config.taf` or by editing the `domain.ini` file directly.

See Also

`domainConfigFile` page 411
`domainScopeKey` page 412

<@DOMDELETE>

Syntax

<@DOMDELETE OBJECT=*variable* [SCOPE=*scope*] [XPOINTER=*Xpointer*] [XPATH=*Xpath*]>

Description

This tag is used to delete XML from a document instance. The OBJECT attribute (and optional SCOPE attribute) defines the variable which contains the document instance. The XPATH / XPOINTER attributes points to the element in the document instance to be deleted. It is not possible to use both of these attributes at once. This tag supports DOM2, XPATH and XPOINTER.

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

Starting with the following document instance in a variable called myDom:

```
<XML>
<DIV>
<P>Paragraph 1</P>
<P>Paragraph 2</P>
</DIV>
</XML>
```

XPointer:

```
<@DOMDELETE OBJECT="myDom"
XPOINTER="child(1).child(2)">
```

XPath:

```
<@DOMDELETE OBJECT="myDom" XPATH="/XML/DIV/
*[position()=2]">
```

deletes part of the XML and results in the following structure in the variable myDom:

```
<XML>
<DIV>
<P>Paragraph 1</P>
</DIV>
</XML>
```

```
<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>
       '>
   ">
```

Using XPATH:

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

Using XPOINTER:

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

See Also

<@DOM>	page 109
<@DOMINSERT>	page 114
<@DOMREPLACE>	page 116
<@DOMSEARCH>	page 118
<@ELEMENTATTRIBUTE>	page 126

<@ELEMENTATTRIBUTES>	page 129
<@ELEMENTNAME>	page 131
<@ELEMENTVALUE>	page 134

<@DOMINSERT>

Syntax

```
<@DOMINSERT OBJECT=variable [SCOPE=scope]
[XPOINTER=Xpointer] [XPATH=Xpath]
[POSITION=append|before|after]>
...XML goes here...</@DOMINSERT>
```

Description

This tag is used to insert XML into a document instance. The `OBJECT` attribute (and optional `SCOPE` attribute) defines the variable which contains the document instance. The `XPOINTER / XPATH` attribute points to an XML element in the document instance.It is not possible to use both of these attributes at once. If the `XPOINTER / XPATH` attribute is omitted, the root element of the document is used. Thus, this tag supports DOM2, XPATH and XPOINTER.

Depending on the value of the `POSITION` attribute, the XML between the start and end tags of `<@DOMINSERT>` is either appended to, put before (that is, a preceding sister), or put after (a following sister) the element specified in `XPATH / XPOINTER`. The default is append.

If the specified variable does not exist, a new variable is created.

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

Starting with the following document instance in a variable called `myDom`:

```
<XML><DIV>
<P>Paragraph 1</P>
<P>Paragraph 2</P>
</DIV>
</XML>
```

XPointer:

```
<@DOMINSERT OBJECT="myDom" XPOINTER="child(1)"
POSITION=append><P>Paragraph 3</P></@DOMINSERT>
```

XPath:

```
<@DOMINSERT OBJECT="myDom" XPATH="/XML/DIV"
POSITION=append><P>Paragraph 3</P></@DOMINSERT>
```

The preceding tag appends the XML between the DOMINSERT tags (<P>Paragraph 3</P>) to the child(1) element (that is, <DIV>). The POSITION attribute is optional in this case, because the default action is to append the XML to the specified element. This results in the following structure:

```
<XML><DIV>
<P>Paragraph 1</P>
<P>Paragraph 2</P>
<P>Paragraph 3</P>
</DIV></XML>
```

The following inserts the specified XML as a preceding sister of the first paragraph:

XPointer:

```
<@DOMINSERT OBJECT="myDom"
XPOINTER="child(1).child(1)"
POSITION=before><P>Paragraph 3</P></@DOMINSERT>
```

XPath:

```
<@DOMINSERT OBJECT="myDom" XPATH="/XML/DIV/
P[position()=1]" POSITION=before><P>Paragraph 3</
P></@DOMINSERT>
```

This results in the following structure:

```
<XML><DIV>
<P>Paragraph 3</P>
<P>Paragraph 1</P>
<P>Paragraph 2</P>
</DIV></XML>
```

See Also

<@DOM>	page 109
<@DOMDELETE>	page 111
<@DOMREPLACE>	page 116
<@DOMSEARCH>	page 118
<@ELEMENTATTRIBUTE>	page 126
<@ELEMENTATTRIBUTES>	page 129
<@ELEMENTNAME>	page 131
<@ELEMENTVALUE>	page 134

<@DOMREPLACE>

Syntax

<@DOMREPLACE OBJECT=*variable* [SCOPE=*scope*] [XPOINTER=*Xpointer*] [XPATH=*Xpath*] ...*XML goes here...*</@DOMREPLACE>

Description

This tag is used to replace XML in a document instance. The OBJECT attribute (and optional SCOPE attribute) defines the variable which contains the document instance. The XPOINTER/XPATH attribute points to the element in the document instance to be replaced. It is not possible to use both attributes at once. Thus, this tag supports DOM2, XPATH and XPOINTER.

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

Starting with the following document instance in a variable called myDom:

```
<XML>
<DIV>
<P>Paragraph 1</P>
<P>Paragraph 2</P>
</DIV>
</XML>
```

XPointer:

```
<@DOMREPLACE OBJECT="myDom"
XPOINTER="child(1).child(2)"><P>A different para.</
P></@DOMREPLACE>
```

XPath:

```
<@DOMREPLACE OBJECT="myDom" XPATH="/XML/DIV/
P[position()=2]"><P>A different para.</P></
@DOMREPLACE>
```

replaces the XML and results in the following structure:

```
<XML>
<DIV>
<P>Paragraph 1</P>
<P>A different para.</P>
</DIV>
</XML>
```

See Also

<@DOM>	page 109
<@DOMDELETE>	page 111
<@DOMINSERT>	page 114
<@DOMSEARCH>	page 118
<@ELEMENTATTRIBUTE>	page 126
<@ELEMENTATTRIBUTES>	page 129
<@ELEMENTNAME>	page 131
<@ELEMENTVALUE>	page 134

<@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 **ELEMENT/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"
```

```
        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'XPOINTER='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>
```

See Also

<@DOM>	page 109
<@DOMDELETE>	page 111
<@DOMINSERT>	page 114
<@DOMREPLACE>	page 116
<@ELEMENTATTRIBUTE>	page 126
<@ELEMENTATTRIBUTES>	page 129
<@ELEMENTNAME>	page 131
<@ELEMENTVALUE>	page 134

<@DQ>, <@SQ>

Description

To use single and double quotes inside a Meta Tag attribute value, use <@SQ> for a single quote "'" and <@DQ> for a double quote "".

Example

```
<@ASSIGN NAME="Important_Quote" VALUE="Yoda said,
<@DQ>Do, or do not; there is no
<@SQ>try<@SQ>.<@DQ>">

<@VAR NAME="Important_Quote">
```

This example returns the following:

```
Yoda said, "Do, or do not; there is no 'try'."
```

<@DSDATE>, <@DSTIME>, <@DSTIMESTAMP>

Syntax

```
<@DSDATE DATE=date [INFORMAT=informat]
[ENCODING=encoding]>
```

```
<@DSTIME TIME=time [INFORMAT=informat]
[ENCODING=encoding]>
```

```
<@DSTIMESTAMP TS=ts [INFORMAT=informat]
[ENCODING=encoding]>
```

Description

These Meta Tags convert a date, time, or timestamp value to the format required by the current action's data source.

The main use for these tags is in Direct DBMS actions. In the other types of database actions (Search, Update, Insert, and Delete), TeraScript performs the required conversion automatically.

The DATE, TIME, and TS attributes are strings in the formats specified by the INFORMAT attribute. This attribute uses the same formatting codes as the date and time formatting configuration variables. If INFORMAT is omitted, the date, time, or timestamp value is assumed to be in the default format, specified by the `dateFormat`, `timeFormat`, and `timestampFormat` configuration variables with system scope, or the current user format, if assigned, using `dateFormat`, `timeFormat`, or `timestampFormat` (user scope).

Note When a two-digit year is given, the following centuries are assumed:

Value	Century
00-36	2000s
37-99	1900s

For example, a two-digit year of 99 is evaluated as `1999`, and a two-digit year of `00` is evaluated as `2000`.

These Meta Tags are valid only in actions associated with a data source.

Example

```
UPDATE myTable SET theDateColumn=<@DSDATE
DATE=<@POSTARG NAME=theDate>>
```

This SQL example from a Direct DBMS action assumes that the date entered by the user into the date form field is in the format specified by `dateFormat`.

See Also

`dateFormat`	page 401
`Encoding Attribute`	page 8
`Format Attribute`	page 11
`timeFormat`	page 401
`timestampFormat`	page 401

<@DSNUM>

Syntax

```
<@DSNUM NUM=num [ENCODING=encoding]>
```

Description

Converts a number to the format required by the current action's data source. The main use for this tag is in Direct DBMS actions. In the other types of database actions (Search, Update, Insert, and Delete), TeraScript performs the required conversion automatically.

This Meta Tag is valid only in actions associated with a data source.

Note Conversion of a number involves removal of thousand separator and currency characters, trimming of spaces from the beginning and end, and substitution of decimal characters with the character required by the DBMS.

Example

```
UPDATE myTable SET theNumericColumn=<@DSNUM
NUM=<@POSTARG NAME=num>>
```

This example assumes the user has entered "$2000.00" into the number form field, and that the system configuration variable `currencyChar` is set to "$", `thousandsChar` is set to "." and that `decimalChar` and `DBDecimalChar` are both set to "."; `<@DSNUM>` tag returns "2000.00".

See Also

`currencyChar`	page 396
`DBDecimalChar`	page 404
`decimalChar`	page 406
`<@DSDATE>`	page 121
`<@DSTIME>`	page 121
`<@DSTIMESTAMP>`	page 121
`Encoding Attribute`	page 8
`thousandsChar`	page 465

<@DSTYPE>

Syntax

```
<@DSTYPE [ENCODING=encoding]>
```

Description

Returns the type of data source associated with the current action. If the current action has no data source associated with it, this tag returns the information for the most recent data source used during the current execution of the application file. If used prior to the execution of a database related action, this tag returns an empty string.

Descriptions of values returned by this Meta Tag are shown in the following table.

Value Returned	Platform(s)	Indicates
FileMaker	Mac OS X	FileMaker Pro
ODBC	All	ODBC
Oracle	All	Native Oracle
JDBC	All	JDBC

Example

```
<@IFEQUAL VALUE1="<@DSTYPE>" VALUE2="ODBC">
display data from an ODBC data source
<@ELSE>
display data from a different data source type
</@IF>
```

This example customizes the HTML returned depending on the data source type.

See Also

<@DBMS> page 98
Encoding Attribute page 8

<@EDITION>

Syntax

```
<@EDITION [ENCODING=encoding]>
```

Description

Returns the edition name of the licensed TeraScript Server.

Example

<@EDITION> returns one of three possible values:

Free

Standard

Advanced

See Also

Encoding Attribute	page 8
<@PLATFORM>	page 221
<@PRODUCT>	page 224
<@VERSION>	page 302

<@ELEMENTATTRIBUTE>

Syntax

<@ELEMENTATTRIBUTE OBJECT=*variable* ATTRIBUTE=*attributename* [SCOPE=*scope*] [XPOINTER=*Xpointer*] [XPATH=*Xpath*] [TYPE=*text*|*array*] [{*array attributes*}]>

Description

This tag is used to return the value of one or more attributes from a document instance.

The OBJECT attribute defines the document instance variable. The SCOPE attribute defines the scope of that document instance variable.

The XPATH/XPOINTER attribute contains a pointer to an element or elements within the document instance. It is not possible to use bother of these attributes at once. This tag supports DOM2, XPATH and XPOINTER.

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 and is aliased to XPOINTER.

The value returned is the value of the attribute defined by NAME. If more than one element is pointed to, and those elements have the attribute defined in ATTRIBUTE, several values may be returned as an array.

There are array-returning attributes that can be used to specify prefixes and suffixes for the returned array, rows within the array, and columns within the rows. They are described in the section Array-to-Text Attributes <page $pagenum>. By default, the returned array is formatted as an HTML table.

If the TYPE attribute is set to TEXT, a returned array is not passed as an array reference when assigning to another variable, but as a text representation of the array, which is returned by default with the array-formatting attributes.

Example

Starting with the following document instance in a variable called myDom:

```
<XML>
<DIV>
<P ID=a111 CLASS=normal>Paragraph 1</P>
```

```
<P ID=b222 CLASS=different>Paragraph 2</P>
</DIV>
</XML>
```

XPointer:

```
<@ELEMENTATTRIBUTE OBJECT="myDom" ATTRIBUTE="ID"
XPOINTER="root().child(1).child(all)">
```

XPath:

```
<@ELEMENTATTRIBUTE OBJECT="myDom" ATTRIBUTE="ID"
XPATH="/XML/DIV/P">
```

returns an array consisting of the two `ID` values:

a111
b222

XPointer:

```
<@ELEMENTATTRIBUTE OBJECT="myDom" ATTRIBUTE="CLASS"
XPOINTER="root().child(1).child(2)">
```

XPath:

```
<@ELEMENTATTRIBUTE OBJECT="myDom" ATTRIBUTE="CLASS"
XPATH="/XML/DIV/P[position()=2]">
```

returns a single value:

```
different
```

```
<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>
        '>
">
```

Using XPATH:

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

Using XPOINTER:

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

See Also

<@DOM>	page 109
<@DOMDELETE>	page 111
<@DOMINSERT>	page 114
<@DOMREPLACE>	page 116
<@DOMSEARCH>	page 118
<@ELEMENTATTRIBUTES>	page 129
<@ELEMENTNAME>	page 131
<@ELEMENTVALUE>	page 134

<@ELEMENTATTRIBUTES>

Syntax

<@ELEMENTATTRIBUTES OBJECT=*variable* [SCOPE=*scope*] [XPOINTER=*Xpointer*] [XPATH=*Xpath*] [TYPE=*text*|*array*] [{*array attributes*}]>

Description

This tag is used to return the value of all attributes of one or more elements from a document instance. The OBJECT attribute defines the document instance variable. The SCOPE attribute defines the scope of that document instance variable.

The XPOINTER/XPATH attribute contains a pointer to an element or elements within the document instance. It is not possible to use both of these attributes at once. All attributes of the element or elements pointed to by XPOINTER/XPATH are returned. Thus, this tag supports DOM2, XPATH and XPOINTER.

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.

There are array-returning attributes that can be used to specify prefixes and suffixes for the returned array, rows within the array, and columns within the rows. They are described in the section Array-to-Text Attributes <page $pagenum>. By default, the returned array is formatted as an HTML table.

If the TYPE attribute is set to TEXT, a returned array is not passed as an array reference when assigning to another variable, but as a text representation of the array, which is returned by default with the array-formatting attributes.

Example

Starting with the following document instance in a variable called myDom:

```
<XML><DIV>
<P ID="a111" CLASS="normal">Paragraph 1</P>
<P ID="b222" CLASS="different">Paragraph 2</P>
</DIV></XML>
```

XPointer:

```
<@ELEMENTATTRIBUTES OBJECT="myDom"
XPOINTER="root().child(1).child(all)">
```

XPath:

```
<@ELEMENTATTRIBUTES OBJECT="myDom" XPATH="/XML/DIV/
*">
```

returns an array consisting of both attribute values:

a111	normal
b222	different

Row 0 (zero) of the array contains the attribute name for each column.

See Also

Array-to-Text Attributes	page 17
<@DOM>	page 109
<@DOMDELETE>	page 111
<@DOMINSERT>	page 114
<@DOMREPLACE>	page 116
<@DOMSEARCH>	page 118
<@ELEMENTATTRIBUTE>	page 126
<@ELEMENTNAME>	page 131
<@ELEMENTVALUE>	page 134

<@ELEMENTNAME>

Syntax

<@ELEMENTNAME OBJECT=*variable* [SCOPE=*scope*] [XPOINTER=*Xpointer*] [XPATH=*Xpath*] [TYPE=*text*|*array*] [{*array attributes*}]>

Description

This tag is used to return an element name or names from a document instance. The OBJECT attribute defines the document instance variable. The SCOPE attribute defines the scope of that document instance variable.

The XPOINTER/XPATH attribute contains a pointer to an element or elements within the document instance. It is not possible to use bother of these attributes at once. The value returned is the name of the element or elements pointed to. If more than one element is pointed to, several values may be returned as an array.

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.

There are array-returning attributes that can be used to specify prefixes and suffixes for the returned array, rows within the array, and columns within the rows. They are described in the section Array-to-Text Attributes <page $pagenum>. By default, the returned array is formatted as an HTML table.

If the TYPE attribute is set to TEXT, a returned array is not passed as an array reference when assigning to another variable, but as a text representation of the array, which is returned by default with the array-formatting attributes.

Example

Starting with the following document instance in a variable called myDom:

```
<XML><DIV>
<P ID=a111 CLASS=normal>Paragraph 1</P>
<P ID=b222 CLASS=different>Paragraph 2</P>
</DIV></XML>
```

XPointer:

```
<@ELEMENTNAME OBJECT="myDom"
XPOINTER="root().child(1).child(all)">
```

XPath:

```
<@ELEMENTNAME OBJECT="myDom" XPATH="/XML/DIV/*">
```

returns a one-dimensional array consisting of both element names:

P
P

XPointer:

```
<@ELEMENTNAME OBJECT="myDom"
XPOINTER="root().child(1).child(2)">
```

XPath:

```
<@ELEMENTNAME OBJECT="myDom" XPATH="/XML/DIV/
*[position()=2]">
```

returns the element name:

```
P
<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>
        '>
">
```

Using XPATH:

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

Using XPOINTER:

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

See Also

Array-to-Text Attributes	page 17
<@DOM>	page 109
<@DOMDELETE>	page 111
<@DOMINSERT>	page 114
<@DOMREPLACE>	page 116
<@DOMSEARCH>	page 118
<@ELEMENTATTRIBUTE>	page 126
<@ELEMENTATTRIBUTES>	page 129
<@ELEMENTVALUE>	page 134

<@ELEMENTVALUE>

Syntax

<@ELEMENTVALUE OBJECT=*variable* [SCOPE=*scope*] [XPOINTER=*Xpointer*] [XPATH=*Xpath*] [TYPE=*text*|*array*] [{*array attributes*}]>

Description

This tag is used to return an element value or values from a document instance. The OBJECT attribute defines the document instance variable. The SCOPE attribute defines the scope of that document instance variable.

The XPOINTER/XPATH attribute contains a pointer to an element or elements within the document instance. It is not possible to use both of these attributes at once.

The value returned is the value of the element or elements pointed to. Other elements that are children of the element are not considered to be content, and are not returned. If more than one element is pointed to, several values may be returned as an array.

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.

There are array-returning attributes that can be used to specify prefixes and suffixes for the returned array, rows within the array, and columns within the rows. They are described in the section Array-to-Text Attributes <page $pagenum>. By default, the returned array is formatted as an HTML table.

If the TYPE attribute is set to TEXT, a returned array is not passed as an array reference when assigning to another variable, but as a text representation of the array, which is returned by default with the array-formatting attributes.

If the specified element has no text content (that is, it is empty, or it contains other elements) then this tag returns an empty string.

Example

Starting with the following document instance in a variable called myDom:

```
<XML><DIV>
<P ID=a111 CLASS=normal>Paragraph 1</P>
<P ID=b222 CLASS=different>Paragraph 2</P>
</DIV></XML>
```

XPointer:

```
<@ELEMENTVALUE OBJECT="myDom"
XPOINTER="root().child(1).child(all)">
```

XPath:

```
<@ELEMENTVALUE OBJECT="myDom" XPATH="/XML/DIV/*">
```

returns a one-dimensional array consisting of both element values:

Paragraph 1
Paragraph 2

XPointer:

```
<@ELEMENTVALUE OBJECT="myDom"
XPOINTER="root().child(1).child(2)">
```

XPath:

```
<@ELEMENTVALUE OBJECT="myDom" XPATH="/XML/DIV/
*[position()=2]">
```

returns the single element value:

```
Paragraph 2
```

```
<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>
        '>
">
```

Using XPATH:

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

Using XPOINTER:

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

See Also

Array-to-Text Attributes	page 17
<@DOM>	page 109
<@DOMDELETE>	page 111
<@DOMINSERT>	page 114
<@DOMREPLACE>	page 116
<@DOMSEARCH>	page 118
<@ELEMENTATTRIBUTE>	page 126
<@ELEMENTATTRIBUTES>	page 129
<@ELEMENTNAME>	page 131

<@EMAIL>

Syntax

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

Description

This tag enables the composition and manipulation of an email message. It is one of the three new tags (<@EMAIL>, <@EMAILSESSION>, and <@MIMEBOUNDARY>) which have been added to allow the user to send and receive email messages using the email protocols SMTP, POP3 and IMAP4.

Three attributes are required:

- **COMMAND** which specifies the function to be executed;
- **NAME** which specifies the mail to be used; and
- **SCOPE** which specifies the scope of the email.

The **PARTID** attribute is the ID for the PART of the email being referenced.

FIELDNAME and **FIELDVALUE** are used in conjunction with the COMMAND 'getfield'.

TYPE has a value of *XML* or *ARRAY* which specifies the structure the message will be stored in. If this attribute is not specified, the default value will be *ARRAY*.

DECODEDATA is an optional attribute which is set to *true* or *false*.

If it is set to true the data being returned will be decoded. If this attribute is not specified, the default value will be *false*.

The **COMMAND** attribute can have any of the values specified in the table below:

Command	Command Function
ADDFIELD	Adds a field to an email (used in conjunction with the STRUCTURE command).
APPENDFIELD	Append a value to a field (used in conjunction with the STRUCTURE command).
EXPORT	Exports an email variable into a text file structured as an email.
GETENTITYBODY	Gets the body of a specified entity
GETFIELD	Gets the field in the email to be returned.
IMPORT	Imports a text file structured as an email into an email variable.
REMOVEFIELD	Removes a field from an email (used in conjunction with the STRUCTURE command).
REPLACEFIELD	Replaces a value of a field (used in conjunction with the STRUCTURE command).
STRUCTURE	Gets the structure of the email.

Whenever any data is returned from the mailserver as type=ARRAY using `<@EMAIL>`, Row 0 of the results array is 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

```
<@EMAIL STRUCTURE NAME=resquest$loopmailvar
SESSIONID='POP3 Sesion:<@USERREFERENCE>'
TYPE=ARRAY>
```

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

```
< @EMAIL GETFIELD NAME=request$loopmailvcar
FIELDNAME=”subject”>
```

```
<@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

See Also

<@EMAILSESSION>	page 140
<@MIMEBOUNDARY>	page 209

<@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 = messageid]
          [NAME = emailname]
          [SCOPE = emailscope]>
```

Description

This tag enables the user to send and receive email messages using the email protocols SMTP, POP3 and IMAP4. It is one of the three new tags (<@EMAIL>, <@EMAILSESSION> and <@MIMEBOUNDARY>).

Three attributes are required:

- **COMMAND** which specifies the function to be executed;
- **PROTOCOL** which specifies the protocol being used to make the connection to the mailserver;
- **SERVER** which specifies the mail server being accessed. This will be either the hostname or the IP address of the machine.

The **SESSIONID** is an optional attribute. It is defined when the OPEN command is used and is thereafter used by the other commands to identify the open session.

The **PORT** attribute is used with the OPEN command. Where not specified it has a default value of 110.

The **USERNAME** attribute is optional. If no username is specified the connection is made anonymously.

The **PASSWORD** attribute is optional. The password will correspond to the username.

The **MAILBOX** attribute is optional. It is used to specify the mail box on the server which should be used.

The **MODE** attribute is only relevant when using the CLOSE command. It is used to either COMMIT or ROLLBACK the changes that have been made to the mail account since the session was opened. If the value is not specified then the default setting will be ROLLBACK which means that the read message will NOT be deleted.

The **FIELDS** attribute is optional. It is used to specify the fields to be used in the chosen command.

The **MESSAGEID** field is only required with the DELETE and RETRIEVE commands. It is used to identify the desire message.

The **NAME** attribute specifies the email name.

The **SCOPE** attribute specifies the email scope.

The **COMMAND** attribute can have any of the values specified in the table below:

Command	Command Function
CLOSE	Closes the email session. This command requires both the SESSIONID and the MODE attributes.
DELETE	Deletes mail from the mailserver. This command requires both the SESSIONID and the MESSAGEID attributes.
LIST	Returns the list of messages currently in the mail account.
OPEN	Opens the email session. To perform an interaction with the mail server this command must be used first. This command requires vlaues for SESSIONID, PROTOCOL and SERVER attributes.
RETRIEVE	Retrieves a specific mail message from the server. This command requires the MESSAGEID attribute.
SEND	Sends a mail that has been contstructed.

Examples

```
<@EMAILSESSION
OPEN
PROTOCOL=”POP3”
```

```
SESSIONID=”POP3 Session:<@USERREFERENCE>”
SERVER=”10.1.2.0” USERNAME=”username”
PASSWORD=”password”
>


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


< @EMAILSESSION
RETRIEVE
NAME=”request$loopmailvar”
MESSAGEID=”<@VAR
request$messagelist[@@request$loopcnt, 1]>”
SESSIONID=”POP3 Session:<@USERREFERENCE>
>
```

See Also

<@EMAIL> page 137
<@MIMEBOUNDARY> page 209

<@ERROR>

Syntax

```
<@ERROR PART=part [ENCODING=encoding]>
```

Description

Returns the value of the named error component specified in the PART attribute of the current error. This Meta Tag is valid only in an action's Error HTML or in an `error.htx` file and is generally used within an <@ERRORS></@ERRORS> block.

For more information, see "defaultErrorFile" on page 408.

The `error.htx` file contains the default HTML to be returned when no Error HTML has been specified for an action or when the error occurs before action execution. Its location is specified by the `defaultErrorFile` configuration variable.

TeraScript may return more than one error at a time, so this Meta Tag should be used inside an <@ERRORS></@ERRORS> block to ensure that the information for all errors generated is shown.

Note In the absence of an <@ERRORS></@ERRORS> block, <@ERROR> returns the first error. However, if an <@ERRORS></@ERRORS> block is found, <@ERROR> tags outside of the block return nothing.

Error Part	Description
CLASS	"Internal" (TeraScript error), "DBMS" (database server error), or "External", (external action error).
APPFILENAME	The file name of the application file that generated the error.
APPFILEPATH	The relative path of the application file that generated the error.
HELPMESSAGE	Allows for the retrieval of a free style optional string describing possible ways to resolve the error.
POSITION	The name of the action that generated the error, if applicable.
NUMBER1	The main error number.
NUMBER2	The secondary error number.
MESSAGE	Allows for the retrieval of a formatted error message.
MESSAGE1	The main error message.
MESSAGE2	The secondary error message.

Example

```
<H3>Error</H3>
An error occurred while processing your request:
<BR>
<@ERRORS>
APPFILE Path:<B><@ERROR PART="APPFILEPATH"></B><BR>
APPFILE Name:<B><@ERROR PART="APPFILENAME"></B><BR>
Position:<B><@ERROR PART="POSITION"></B><BR>
Class:<B><@ERROR PART="CLASS"></B><BR>
Main Error Number: <B><@ERROR PART="NUMBER1"></B>
<BR>
</@ERRORS>
```

This example returns all of the error information for each error encountered during the current action execution.

See Also

`defaultErrorFile`	page 408
Encoding Attribute	page 8
`<@ERRORS> </@ERRORS>`	page 145

<@ERRORS> </@ERRORS>

Description

If more than one error occurs during application file execution, TeraScript Server queues up the errors. <@ERRORS>, in conjunction with <@ERROR>, allows you to iterate over the list of errors. If the <@ERRORS></@ERRORS> block is not used, information about the first error encountered is returned by <@ERROR>.

Text between these tags is processed for each error generated by the associated action. The tags are valid only in an action's Error HTML or in an `error.htx file`.

The `error.htx file` contains the default HTML to be returned when no Error HTML has been specified for an action or when the error occurs before action execution. Its location is specified by the `defaultErrorFile` configuration variable.

Example

```
<H3>Error</H3>
An error occurred while processing your request:
<BR>
<@ERRORS>
Position: <B><@ERROR PART=POSITION></B><BR>
Class: <B><@ERROR PART=CLASS></B><BR>
Main Error Number: <B><@ERROR PART=NUMBER1></B><BR>
Secondary Error Number: <B><@ERROR PART=NUMBER2></B>
<BR>
Main Error Message: <B><@ERROR PART=MESSAGE1></B>
<BR>
Secondary Error Message: <B><@ERROR PART=MESSAGE2>
</B><BR>
</@ERRORS>
```

This example returns all of the error information for each error encountered during the current action execution.

See Also

`defaultErrorFile` page 408
<@EMAIL> page 137

<@EXCLUDE> </@EXCLUDE>

Syntax

<@EXCLUDE>*text*</@EXCLUDE>

Description

Processes *text* for Meta Tags, without adding the results of that processing to the Results HTML.

Like the <@COMMENT></@COMMENT> Meta Tag block, the contents of the <@EXCLUDE></@EXCLUDE> Meta Tag block are removed from the HTML sent on to the Web server. Unlike that tag block, the contents of the <@EXCLUDE></@EXCLUDE> block are processed, and Meta Tags are executed, before the contents of the <@EXCLUDE> are removed from the Results HMTL.

This tag is useful if you want to do processing in Results HTML without adding empty lines to the HTML returned.

Note You must use both a start tag and an end tag when using <@EXCLUDE>. Unpaired appearances are treated as unrecognized tags and left untouched.

Example

```
<@EXCLUDE>Do this: <@ASSIGN NAME=myVar
VALUE="asdfasd"></@EXCLUDE>
```

The tag pair and the text contained inside ("Do this:") are removed before the HTML is returned, but the <@ASSIGN> Meta Tag is executed as part of the application.

See Also

<@COMMENT> </@COMMENT> page 77

<@EXIT>

Description

Causes processing of the current Results HTML, No Results HTML, or Error HTML to end. Processing of the application file continues with the next action. This tag has no attributes.

This tag is generally used with an <@IF> tag to terminate processing of the current HTML when some condition is met.

Example

```
[...standard results are found here...]
<@IF EXPR="@@user$accesslevel>5" FALSE=<@EXIT>>
Here are some additional details on the records that
were returned:
<@ROWS>
<STRONG>Name:</STRONG> <@COLUMN
NAME="user.name"><BR>
<STRONG>Password:</STRONG> <@COLUMN
NAME="user.password"><BR>
</@ROWS>
```

This example processes the block of Results HTML only if the user has privileges on the system, that is, if the user's access level is greater than "5".

See Also

<@BREAK> page 42
<@CONTINUE> page 82

<@FILTER>

Syntax

<@FILTER ARRAY=*arrayVarName* EXPR=*filterExpr* [SCOPE=*scope*] [{*array attributes*}]>

Description

Given an array, this Meta Tag returns an array containing rows matching a specified expression. The ARRAY attribute specifies the name of a variable containing an array. The EXPR attribute specifies the expression to use when evaluating each row to determine whether it will be in the array returned. In this expression, the values from the current row are specified with a number sign (#), followed by the column name or number. (See the examples following.) This expression may use any operators and functions supported by the <@CALC> tag. If the expression evaluates to 1 (true) for a particular row, that row appears in the output array.

The SCOPE attribute specifies the scope of the variable specified in the value of the ARRAY attribute. If SCOPE is not specified, the default scoping rules are used.

There are array-returning attributes that can be used to specify prefixes and suffixes for the returned array, rows within the array, and columns within the rows. They are described in the section Array-to-Text Attributes <page $pagenum>. By default, the returned array is formatted as an HTML table.

Meta Tags are permitted in any of the attributes, but see the following note. Meta Tags specified in EXPR are evaluated for each row in ARRAY.

Note References to columns inside the EXPR attribute cannot be specified by Meta Tags.

Examples

- Assume the request variable resultSet contains the following array:

3243	Acme Insurance	ACTIVE
2344	Fairview Electronics	INACTIVE
2435	Vanguard Computing	INACTIVE
1234	Cinetopia	ACTIVE
5421	Trailblazer Industries	ACTIVE

<@FILTER ARRAY="resultSet" SCOPE="request" EXPR="#3=ACTIVE"> returns:

3243	Acme Insurance	ACTIVE
1234	Cinetopia	ACTIVE
5421	Trailblazer Industries	ACTIVE

- Assume the user variable `orders` contains the following array and that column two is named `amount` and column three is named `state`:

1000	324.78	NY
1001	849.25	MA
1002	1245.97	CT
1003	400.45	CA
1004	598.10	NY
1005	53.89	ME
1006	1800.76	NY

<@FILTER ARRAY="orders" SCOPE="user" EXPR="(#amount > 500) and (#state = NY)"> returns:

1004	598.10	NY
1006	1800.76	NY

- Assume the user variable `accounts` contains the following array and that column two is named `credit` and column three is named `debit`:

987235-2347	3257.65	2049.12
324234-9848	5234.37	6097.90
234349-2823	0.00	56.33
630780-8491	657.78	347.20
324969-1983	234561.27	229679.18
196573-8436	326.62	192.20
537030-4739	9482.40	10274.23

Also assume the value `-100` is stored in the variable `od_limit`.

<@FILTER ARRAY="accounts" SCOPE="user" EXPR="(#credit - #debit) < @@od_limit"> returns:

324234-9848	5234.37	6097.90
537030-4739	9482.40	10274.23

See Also

Array-to-Text Attributes	page 17
<@ADDROWS>	page 20
<@DELROWS>	page 103
<@DISTINCT>	page 105
<@INTERSECT>	page 171
<@SORT>	page 256
<@UNION>	page 277

<@FOR> </@FOR>

Syntax

```
<@FOR [START=start] [STOP=stop] [STEP=step]>
</@FOR>
```

Description

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

<@FOR> 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 as many times as the <@FOR> loop specifies. The start and stop values can be specified, as can the step used to get from one to the other.

Inside a for loop, <@CURROW> can be used to get the value of the index.

START defines the starting value for the index, for which the default value is "1".

STOP defines the stopping value for the index. The loop terminates when this value is exceeded, not when it is reached. The default value is "0".

STEP defines the increment added to the index after each iteration. The default value is "1".

PUSH, a deprecated attribute [PUSH=*push*], allows the sending of data. This attribute has been deprecated as of the release of TeraScript 6. It is currently operational but will be removed in the next major version release. Developers are encouraged to discontinue use of this attribute. When applicable, a warning will be reported to the `witangoevents.log` file.

This tag must appear in pairs and cannot span multiple actions. If the specified step cannot take the index from start to stop, no iterations are made. If the start equals the stop, one iteration is made, regardless of the step size.

Nested <@FOR> loops are supported. The <@CURROW> Meta Tag will display the index for the proper loop where it is located.

Example

```
<@FOR STOP="5">
This function does this (Loop: <@CURROW>) <BR>
</@FOR>
```

This example outputs the following:

```
This function does this (Loop: 1)
This function does this (Loop: 2)
This function does this (Loop: 3)
This function does this (Loop: 4)
This function does this (Loop: 5)
```

See Also

`<@CURROW>` page 89
`<@WHILE> </@WHILE>` page 304

<@FORMAT>

Syntax

<@FORMAT STR=*string* [FORMAT=*format*] [INFORMAT=*informat*] [ENCODING=*encoding*]>

Description

Allows access to the reformatting routines independent of the other tags. The tag takes a STR attribute for the text to reformat and an optional FORMAT attribute indicating the desired output format. An optional INFORMAT attribute is provided for datetime-class formatting to accept non-standard datetime values.

Examples

```
<@FORMAT STR="<@CURRENTTIMESTAMP>"
FORMAT="datetime:%Y-%m-%d" INFORMAT="datetime:<@VAR
NAME='timestampFormat'>">
```

This example demonstrates how to output the current date in ODBC/ISO style, purposely using a timestamp.

```
If a kilobyte is 1024 (2^10 bytes), then a megabyte
should be <@FORMAT STR=<@CALC EXPR="1024 * 1024">
FORMAT="num:comma-integer"> bytes.
```

This example demonstrates how to output a thousands-grouped integer value.

See Also

Encoding Attribute page 8
Format Attribute page 11

<@GETPARAM>

Syntax

<@GETPARAM NAME=*name* [TYPE=*text*] [ENCODING=*encoding*] [FORMAT=*format*] [{*array attributes*}]>

Description

<@GETPARAM> retrieves the contents of a parameter variable within a TeraScript class file. This tag is similar to <@VAR>, but performs error checking to ensure that only parameters of a TeraScript class file (which must be in method scope) can be retrieved.

This Meta Tag is specifically used for retrieving the value of a parameter in a TeraScript class file. If the variable specified by the NAME attribute is not a TeraScript class file parameter, this tag returns an error.

Note Because the parameter variables specified by <@GETPARAM> are only valid in method scope, scope cannot be specified in the NAME attribute, unlike the <@VAR> Meta Tag (for example, NAME=`request$foo` generates incorrect results).

This tag is only valid within a TeraScript class file method.

Text

When retrieving the contents of a text variable (standard variable), the result of <@GETPARAM> is always a text string.

Arrays

<@GETPARAM> may also be used to retrieve an array. However, <@GETPARAM> does different things to arrays based on context: <@GETPARAM> converts the array to text whenever the result of the tag is returned in Results HTML, or when TYPE=text is specified; <@GETPARAM> returns an internal reference to the array when it is used to copy an array from one place to another. So, if <@GETPARAM> is used within <@ASSIGN>, then no conversion to text is performed (unless the TYPE="text" attribute is specified).

There are array-returning attributes that can be used to specify prefixes and suffixes for the returned array, rows within the array, and columns within the rows. They are described in the section `Array-to-Text Attributes  <page $pagenum>`. By default, the returned array is formatted as an HTML table.

Example

Within the return HTML of a TeraScript class file method, you could use the following series of Meta Tags to get the value of an In parameter (in this case, the radius of a sphere), perform calculations on it (calculating the surface area of a sphere), and set the value of a returned (Out) parameter accurate to two decimal places:

```
<@SETPARAM NAME=OutSurface VALUE=<@CALC
EXPR="4*P*(<@GETPARAM NAME=Radius>^2)"
PRECISION=2>>
```

See Also

Array-to-Text Attributespage 17
Encoding Attribute page 8
Format Attribute page 11
<@SETPARAM> page 253

<@HTTPATTRIBUTE>

Syntax

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

Description

Evaluates to the value of the specified HTTP attribute. HTTP attributes are values passed to TeraScript 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 is 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.
FULLHEADER	Sends the full header to the TeraScript Server.
FROM_USER	Rarely returns anything; with some older Web browser applications, the user's e-mail address.
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 TeraScript.)
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".

NAME attribute values	Description

PATH_ARGS	Text after the base URL (which includes the TeraScript 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/TeraScript-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.
SOAPAction	Returns the SOAPAction value is it is specified in the HTTP header. This value is only relevant to SOAP calls made to the 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.

NAME attribute values	Description
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.

Note This Meta Tag was previously known as <@CGIPARAM>, it is now an alias of <@HTTPATTRIBUTE>.

See Also

Encoding Attribute page 8

<@HTTPREASONPHRASE>

Syntax

```
<@HTTPREASONPHRASE>
```

Description

The primary use of this tag is in the default header returned by the TeraScript application server. This tag indicates the status of the web page being generated. It is used in conjunction with <@HTTPSTATUSCODE> to form a proper HTTP Response header.

<@HTTPREASONPHRASE> reports the matching status reason phrase: OK or Application Server Error.

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 ...
```

See Also

<@HTTPSTATUSCODE> page 160
<@SETCOOKIES> page 252

<@HTTPSTATUSCODE>

Syntax

```
<@HTTPSTATUSCODE>
```

Description

The primary use of this tag is in the default header returned by the TeraScript application server. This tag indicates the status of the web page being generated. It is used in conjunction with <@HTTPREASONPHRASE> to form a proper HTTP Response header.

<@HTTPSTATUSCODE> evaluates to either 200 which indicates that the page is without error, or, 500 which indicates that the page does have problems.

See Also

<@HTTPATTRIBUTE> page 156
<@SETCOOKIES> page 252

<@IF>, <@ELSEIF>, <@ELSEIFEMPTY>, <@ELSEIFNOTEMPTY>,<@ELSEIFEQUAL>, </@IF>

Syntax

The <@IF> Meta Tag takes one of two forms:

Form One

```
<@IF EXPR=expr [TRUE=true] [FALSE=false]>
```

Form Two

```
<@IF EXPR=expr>
        ifText
[<@ELSEIF EXPR=expr>
        elseIfText]
[<@ELESIFEMPTY VALUE=value>
        elseIfEmptyText]
[<@ELESIFNOTEMPTY VALUE=value>
        trueSubsitutionText]
[<@ELSEIFEQUAL VALUE1=value1 VALUE2=value2>
        elseIfEqualText]
[<@ELSE>
        elseText]
</@IF>
```

Description

Both forms of the <@IF> Meta Tag take EXPR attributes. The expression specified is evaluated just like the EXPR attribute of the <@CALC> Meta Tag, and all of the operations permitted in it are permitted here.

The EXPR attribute value must be quoted. The expression is evaluated as false if it returns "false" or "0" (zero); otherwise, the expression is considered to be true.

For more information, see "<@CALC>" on page 43.

Expressions can be of any degree of complexity and they are processed according to <@CALC> grammar; that is, you can use parentheses to order expressions, logical functions such as AND and OR, and string or numeric functions such as len(), sin(), or max().

For example, the following complex expression is valid as the value of the EXPR attribute:

```
<@IF EXPR="(len(@@password) > 6) OR (len(@@password)
< 3)" TRUE="Passwords must have between 3 and 6
characters. Try again." FALSE="That's a valid
password.">
```

This example checks the length of the `password` variable to see if it is between three and six characters and returns different text if the expression evaluates to true or false.

Form One

This form of the `<@IF>` Meta Tag returns one of two values based on the evaluation of `EXPR`. If the expression is true, the value specified in the `TRUE` attribute is returned. If the expression is false, the value specified in the `FALSE` attribute is returned.

This form of the `<@IF>` Meta Tag may be used anywhere that a value-returning Meta Tag is permitted.

Form Two

This form of the `<@IF>` Meta Tag processes blocks (of text, HTML, SQL) depending on the evaluation of the `EXPR` attribute. If the expression is true, the text after the tag—up until an ending `</@IF>`— is processed.

The `<@ELSE>` Meta Tag and its variations (`<@ELSEIF>`, `<@ELSEIFEMPTY>`, and `<@ELSEIFEQUAL>`) can be used inside of an `<@IF></@IF>` block to provide alternate expressions and corresponding text blocks to be processed if the `<@IF>` tag's expression is false.

The `<@ELSE>` Meta Tag takes no attributes. The text block associated with it is processed and then processing of the enclosing `IF` block ends.

The other `ELSE` tags are conditional. Their text blocks are processed only if the condition specified is met.

For more information, see "<@IFEMPTY> <@ELSE> </@IF>" on page 165 and `<@IFEQUAL> <@ELSE> </@IF> <page $pagenum>` for descriptions of how the `<@ELSEIFEQUAL>` and `<@ELSEIFEMPTY>` conditions are evaluated.

The `<@ELSEIF>` tag's expression is evaluated just like the `<@IF>` tag's expression. Once an `ELSE` condition is met, the text block associated with it is processed and then processing of the enclosing if block ends. If an `ELSE` condition is not met, processing continues with the next `ELSE` tag in the `IF` block.

Any number of `<@ELSEIF>` tags may be used inside an `<@IF> </@IF>` block.

<@IF>, <@IFEMPTY>, and <@IFEQUAL> Meta Tag blocks may be nested; that is, the text block associated with an IF or ELSE block may itself contain an if block.

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.

This second form of the <@IF> Meta Tag may be used only in HTML windows, Direct DBMS action SQL, and in the text of scripts for the Script action.

Examples

```
<@IF EXPR="<@VAR CD>='ABBA'" TRUE="Cool!" FALSE="Too
Bad">
```

Evaluates to "Cool!" if the CD variable is equal to the text ABBA; otherwise, returns "Too Bad".

```
<@IF EXPR="<@CURRENTTIME FORMAT='%H'> <4 &&
<@CURRENTTIME FORMAT='%H'>> 0">
       Wow, you're up late!
</@IF>
```

Displays "Wow, you're up late!" if the current time is between 1:00 AM and 3:59 AM.

```
<@IF EXPR="<@VAR NAME='choice'>=1">
       first choice HTML
<@ELSEIF EXPR="<@VAR NAME='choice'>=2">
       second choice HTML
<@ELSEIF EXPR="<@VAR NAME='choice'>=3">
       third choice HTML
<@ELSE>
       default choice HTML
</@IF>
```

This example displays different HTML based on the value of the choice variable. If it evaluates to "1", "first choice HTML" is displayed; if it evaluates to "2", "second choice HTML" is displayed; and so on. If it does not evaluate to "1", "2", or "3", "default choice HTML" is displayed.

There is a shortcut syntax for returning variables as well, with or without scope: use a double "@" and the name of the variable. The following two notations are equivalent: <@VAR NAME="homer"> *or* @@homer

```
<@IF EXPR="@@category=color">
       <@IF EXPR="@@color=red">
       Fire engines, apples, and embarrassed
       faces come in this color.
       <@ELSEIF EXPR="@@color=blue">
       Ah, the color of clear skies, the ocean,
       and recycling boxes.
       <@ELSE>
       I'm sure that's a fine hue, but I know
       nothing about it.
       </@IF>
<@ELSEIF EXPR="@@category=shape">
       <@IF EXPR="@@shape=circle">
       Reminds me of the moon, clock faces, and
       my old LPs.
       <@ELSEIF EXPR="@@shape=triangle">
       Yield signs, slices of hot apple pie, and
       dog ears have this form.
       <@ELSE>
       Hmm. The shape of things to come, perhaps?
       </@IF>
<@ELSE>
Colors and shapes are my only areas of expertise.
</@IF>
```

This example demonstrates nested ifs. The outer if block checks for the category. Inside the block for each category, a nested if block checks for particular values in the category.

See Also

<@CALC> page 43
<@IFEMPTY> <@ELSE> </@IF> page 165
<@IFEQUAL> <@ELSE> </@IF> page 166
<@IFNOTEMPTY> <@ELSE> </@IF>page 168

<@IFEMPTY> <@ELSE> </@IF>

Syntax

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

Description

If the value specified in VALUE is an empty string, <@IFEMPTY VALUE=*value*><@ELSE></@IF> includes *trueSubstitutionText*; otherwise, it includes *falseSubstitutionText*. The VALUE attribute value may be a Meta Tag or literal value (though it makes little sense to use a literal value). The <@ELSE> portion is optional.

The *trueSubstitutionText* and *falseSubstitutionText* may include other <@IF>,<@IFEMPTY>, and <@IFEQUAL> Meta Tags.

Example

```
<@IFEMPTY VALUE="<@HTTPATTRIBUTE NAME='USERNAME'>">
       Here are the guest options:
       ...guest options...
<@ELSE>
       <@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>
</@IF>
```

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

See Also

<@CALC> page 43
<@IF>, <@ELSEIF>, <@ELSEIFEMPTY>,
<@ELSEIFNOTEMPTY>,<@ELSEIFEQUAL>, </@IF>page 161
<@IFEQUAL> <@ELSE> </@IF> page 166
<@IFNOTEMPTY> <@ELSE> </@IF>page 168

<@IFEQUAL> <@ELSE> </@IF>

Syntax

```
<@IFEQUAL VALUE1=value1 VALUE2=value2>
      trueSubstitutionText
[<@ELSE>
      falseSubstitutionText]
</@IF>
```

Description

If the value of the VALUE1 attribute and the value of the VALUE2 attribute are equal, <@IFEQUAL> includes *trueSubstitutionText*; otherwise it includes *falseSubstitutionText*. Each of the attributes may be a Meta Tag or a literal value, or a combination of both. Literal values must be quoted if they contain a space. The <@ELSE> portion is optional.

<@IFEQUAL> can be used to do *begins-with* type comparisons. An asterisk at the end of either value acts as a wildcard character, matching any characters at the end of the other value attribute. (You can search for an asterisk character by using <@CHAR 42>.)

When comparing the values, TeraScript attempts to convert both values to numbers and perform a numeric comparison. If one or both values cannot be converted to numbers, TeraScript performs a string comparison.

The *trueSubstitutionText* and *falseSubstitutionText* may include other <@IF>, <@IFEMPTY>, and <@IFEQUAL>.

Examples

```
<@IFEQUAL VALUE1="<@HTTPATTRIBUTE
NAME='user_agent'>"
VALUE2="Mozilla*">
...HTML for Netscape Navigator...
<@ELSE>
...HTML for other Web browsers...
</@IF>
```

This example returns different HTML depending on the user's Web browser.

```
<SELECT NAME="region">

<OPTION VALUE="NE"
<@IFEQUAL VALUE1="<@COLUMN 'customer.region'>"
VALUE2="NE">SELECTED</@IF>>North East
```

```
<OPTION VALUE="NW"
<@IFEQUAL VALUE1=<@COLUMN
customer.region>VALUE2"NW">SELECTED</@IF>>North
West

<OPTION VALUE="SE" <@IFEQUAL VALUE1=<@COLUMN
customer.region>VALUE2="SE">SELECTED </@IF>>South
East

<OPTION VALUE="SW" <@IFEQUAL VALUE1=<@COLUMN
customer.region>
VALUE2="SW">SELECTED</@IF>>South West

</SELECT>
```

This example sets the correct pop-up menu item to `SELECTED` based on the value of a database field.

See Also

`<@CALC>` page 43
`<@IFEMPTY> <@ELSE> </@IF>` page 165
`<@IF>, <@ELSEIF>, <@ELSEIFEMPTY>, <@ELSEIFNOTEMPTY>,<@ELSEIFEQUAL>, </@IF>` page 161
`<@IFNOTEMPTY> <@ELSE> </@IF>` page 168

<@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">.

See Also

<@CALC> page 43
<@IFEMPTY> <@ELSE> </@IF> page 165
<@IF>, <@ELSEIF>, <@ELSEIFEMPTY>,

<@ELSEIFNOTEMPTY>,<@ELSEIFEQUAL>, </@IF>
page 161
<@IFEQUAL> <@ELSE> </@IF> page 166

<@INCLUDE>

Syntax

<@INCLUDE FILE=*file*>

Description

Returns the contents of the specified file. The file may contain Meta Tags, which are processed normally. The FILE attribute is a *slash-separated* path from the Web server root. The FILE attribute may include literal text, Meta Tags, or both.

If TeraScript cannot find the referenced file, the Meta Tag returns an empty value. This Meta Tag may be used in Results, No Results and Error HTML, Direct DBMS SQL, variable assignment values, External action attributes, and in database action insert, update, and criteria value fields.

Examples

```
<@INCLUDE FILE="/Footers/my_footer.html">
```

This example includes the `my_footer.html` file residing in the Footers directory in the TeraScript application file root directory.

```
<@INCLUDE FILE="<@APPFILEPATH>my_footer.html">
```

This example includes the `my_footer.html` file residing in the same directory as the currently executing application file.

```
<@INCLUDE FILE="<@COLUMN NAME='invoice.filename'>">
```

This example includes the contents of the file specified in the `filename` column in the invoice table.

<@INTERSECT>

Syntax

```
<@INTERSECT ARRAY1=arrayVarName1 ARRAY2=arrayVarName2
[COLS=compCol [compType] [, ...]] [SCOPE1=scope1]
[SCOPE2=scope2] [{array attributes}]>
```

Description

Returns the intersection of two arrays, that is, an array containing only those rows that exist in both input arrays.

The two input arrays are not modified. To store the result of this Meta Tag in a variable, use a variable assignment.

The ARRAY1 and ARRAY2 attributes specify the names of variables containing arrays. The optional COLS attribute specifies the column(s) to consider when determining whether two rows are the same: the columns are specified using column numbers or names (*compCol*), with an optional comparison type (*compType*). The arrays must have the same number of columns; otherwise, an error is generated.

Valid comparison types are SMART (the default), DICT, ALPHA and NUM. DICT compares columns alphabetically, irrespective of case. ALPHA performs a case-sensitive comparison. NUM compares columns numerically. SMART checks whether values are numeric or alphabetic and performs a NUM or DICT comparison.

If no COLS attribute is specified, the intersection of the two arrays is accomplished via a SMART comparison type that examines all columns.

The SCOPE1 and SCOPE2 attributes specify the scope of the variables specified by ARRAY1 and ARRAY2, respectively. If the attribute is not specified, the default scoping rules are used.

There are array-returning attributes that can be used to specify prefixes and suffixes for the returned array, rows within the array, and columns within the rows. They are described in the section Array-to-Text Attributes <page $pagenum>. By default, the returned array is formatted as an HTML table.

Meta Tags are permitted in any of the attributes.

Examples

- If the variable p_items contains the following array:

red
blue

green
orange

The variable new contains the following array:

orange
pink
blue
pink

<@INTERSECT ARRAY1="p_items" ARRAY2="new"> returns:

blue
orange

- If the variable test contains:

1	a	a
2	b	c
3	c	c
4	b	c

and the variable test2 contains:

1	a	a
2	b	b
3	c	c

<@INTERSECT ARRAY1="test" ARRAY2="test2"> returns:

1	a	a
3	c	c

- The variable usr1 contains the following array:

Gilbert	Steve	1823-1344	$433.00
Brown	Robert	5543-1233	$332.50
Brown	Marsha	1122-5778	$541.00

The variable usr2 contains the following array:

Kelly	Herbert	5543-1443	$100.50
Brown	Robert	6670-1123	$1123.75

To find users that appear in both arrays, you would find the intersection of the two arrays based on the first two columns:
`<@INTERSECT ARRAY1="usr1" ARRAY2="usr2" COLS="1, 2">`
returns:

Brown	Robert	6670-1123	$1123.75	*

* TeraScript returns just one of the rows that have the same values in the specified columns (1 and 2).

Only columns 1 and 2 are specified as relevant; the different values in the other columns are ignored for the purposes of comparison.

- In conjunction with `<@IF>`, `<@INTERSECT>` may be used to test for the existence of a row in another array. If `Var_A` contains the following array:

1	John	Tesh	A
2	Mary	Hart	B
3	Bob	Mackie	C
4	Sharon	Tate	D

`Var_B` contains the following array:

3	Bob	Mackie	C

```
<@IF EXPR="<@INTERSECT Var_A Var_B>">
   Var_B is in Var_A
<@ELSE>
   Not in Var_A
</@IF>
```

This is because an array value specified as an expression (in `<@CALC>` or `<@IF>`) returns the number of rows in that array.

For more information, see `Array evaluation <page $pagenum>`.

See Also

Array-to-Text Attributes	page 17
`<@DISTINCT>`	page 105
`<@FILTER>`	page 148
`<@SORT>`	page 256
`<@UNION>`	page 277

<@ISALPHA>

Syntax

<@ISALPHA STR=*mystring*>

Description

Evaluates to non-zero if the expression specified in STR is an contains only alphabetic characters (that is, A-Z and a-z).

An empty or blank expression is not considered a string.

Examples

<@ISALPHA STR="abcdefg"> *true*

<@ISALPHA STR="1"> *false*

See Also

<@ISDATE> page 176
<@ISTIME> page 176
<@ISTIMESTAMP> page 176
<@ISNUM> page 184

<@ISALPHANUM>

Syntax

```
<@ISALPHANUM [STR=]string>
```

Description

Evaluates to "1" (true) if the string provided consists of letters (of either case) and/or digits and no other characters. All letters of the Alphabet are valid (a-z) as well as their capitals (A-Z) and all numbers (0-9). It is not necessary for there to be both a letter and number to produce a true result.

Example

```
<@ISALPHANUM STR="1234">
```

Evaluates to "1" (true)

```
<@ISALPHANUM STR="ASDF">
```

Evaluates to "1" (true)

```
<@ISALPHANUM STR="MyPa22W0rD">
```

Evaluates to "1" (true)

```
<@ISALPHANUM STR="My-P@$$w0RD!">
```

Evaluates to "0" (false)

See Also

<@ISALPHA>	page 174
<@ISDECIMAL>	page 180
<@ISINT>	page 181
<@ISNUM>	page 184
currencyChar	page 396
decimalChar	page 406
thousandsChar	page 465

<@ISDATE>, <@ISTIME>, <@ISTIMESTAMP>

Syntax

```
<@ISDATE VALUE=date>
<@ISTIME VALUE=time>
<@ISTIMESTAMP VALUE=timestamp>
```

Description

These tags attempt to parse the input value and see if it is a valid date, time, or timestamp, respectively. The intent of the tags is to detect as wide a variety of formats as possible, thus allowing users greater choice in inputting values. The tags evaluate to the value "1" or "0".

If the value contains spaces, it must be quoted (single or double, as appropriate).

The tags currently support the following date/time/timestamp formats:

- configuration variable defaults
- ISO 8601 formats (complete representations only)
- ODBC formats
- numeric formats
- textual formats.

All formats assume the Gregorian calendar; that is, they use Gregorian rules for all time periods as opposed to switching back to the Julian calendar for years before the adoption of the Gregorian calendar, which may vary depending on the country. All years must be greater than zero.

A date unacceptable in one format may be acceptable in another. For example, 98-02-12 is not a valid ODBC nor ISO date, but is detected as a general numeric date because it is sufficiently unambiguous.

ISO Date Format

There are three ways to specify a date in ISO format:

- **Calendar Date Format: yyyy-mm-dd**. An ISO Calendar Date format gives years, months, and dates in numeric values. All digit places of each field must be filled. Use leading zeroes to pad fields to full width. Hyphens are optional, but if present, all

must be present; they are all-or-none optional, for example, "1998-05-01" or "19980501".

- **Week/Day Format: yyyy-Www-d**. An ISO Week/Day format specifies a date with its week number in the given year, plus its day in the week. The capital W is required, the hyphens are all-or-none optional, and numbers must be full-width. Weeks range from W01 to W53, and days in each week are numbered one (Monday) to seven (Sunday).

  Week W01 of any year is defined as the first week with the majority of the days of that week in that year; for example, it is the week that January first is in if January first falls on a Monday to a Thursday, or else it is the next week. Alternately, the week containing January 04 is W01. Remember that ISO defines a week as Monday to Sunday.

Note Note that the calendar year may be different from the week year. For example, 1998-W01-2=1997-12-31, is December 31, 1997.

- **Ordinal Date Format: yyyy-ddd**. An ISO Ordinal Date format specifies a year and the day in that year numbered from January first as 001. The day number ranges from 001 to 365 (366 in leap years). The hyphen is optional. The full width of the digit fields must be provided; use leading zeroes to pad fields.

ISO Time Format

An ISO time is specified in a 24-hour clock format: **hh:mm:ss**

The string may be preceded by a capital T, and may have a decimal fraction portion consisting of a comma or period followed by one to nine digits. Colons are all-or-none optional.

Note ISO allows 24:00 to indicate 00:00:00 on the next day, but TeraScript does not allow this.

ISO Timestamp Format

An ISO timestamp format is simply the concatenation of a date and a time in that order, with the capital T before the time mandatory. Again, no spaces ever appear in an ISO format, for example, "1998-05-01T12:00:00".

ODBC Formats

ODBC date/time string formats are very strict. No special interpretation is required.

ODBC Date Format

Dates are specified yyyy-mm-dd using calendar dates. The full width of each field must be provided. Use leading zeroes to pad fields to full width. Hyphens are required.

ODBC Time Format

The time format hh:mm:ss.fffffffff is a triple of two-digit numbers representing a 24-hour time, with colons required, followed by an optional fraction portion consisting of a period with one to nine decimal digits afterwards. Use leading zeroes to pad fields to full width.

ODBC Timestamp Format

Timestamps are made by specifying a date, followed by a single space, followed by the time.

Numeric Formats

A numeric format is defined to be a date or time specified fully by using numbers, separating punctuation, and possibly an "AM" or "pm" marker. Any strings with words inside fall into the *Textual* category.

These tags do not attempt to resolve ambiguities according to the current locale or TeraScript Server settings. Ambiguous values are not accepted.

Dates are composed of three numbers separated by identical punctuation character sequences: "/", "//", ".", or "-". Times are specified by three numbers separated by identical punctuation characters: ":" or ".", with an optional am/pm (case insensitive) marker afterwards. If an am/pm marker is present, then a single space may separate it and the time numbers. Timestamps are created by writing a date, followed by white space, followed by a time. A time may never be specified first.

Textual Formats

A textual format is any date/time string that includes alphabetic characters. These words are assumed to be weekday and month names in a variety of different languages. Input text must use high-ASCII characters instead of HTML `&#xxx;` escapes to represent

accented characters. The following languages that use the ISO-Latin-1 character coding set are supported:

- C/POSIX DEFAULT
- Danish
- German
- English
- Spanish
- Finnish
- French
- Icelandic
- Italian
- Dutch
- Norwegian
- Portuguese
- Swedish.

A date may be written in any of the following formats, with [] indicating optional items.

- [weekday] month day year
- [weekday] day month year
- year month day (hyphen delimiters allowed).

The weekday may only be followed by an optional comma and a space. Other items may use dots, or single dots as well. If a weekday is specified, it must be correct. For example, June 13 1997 was a Friday, and anything else is wrong. No extraneous words should appear in the string, such as the "de" in Spanish "viernes, 20 de junio de 1997". In general, no punctuation is best. (Punctuation is supported to the point of allowing what is commonly in use today.) All word comparisons are case-insensitive.

If a time is given, it must have three numbers, two digits long (1–2 for the hour), separated by "." or ":", and an optional space with an optional am/pm marker used in that native language. No delimiters follow or precede a time otherwise. A time may appear anywhere in the text.

Note The current implementation of the IS[DATE/TIME/TIMESTAMP] tags only works with languages that use the ISO-Latin-1 character set.

<@ISDECIMAL>

Syntax

```
<@ISDECIMAL [STR=]string>
```

Description

Evaluates to "1" (true) if the string provided is a number and contains a decimal character. The evaluation of the string as a number is the same test done by <@ISNUM>, therefore the currency character and the thousands character are allowed. The evaluation of this tag differs from <@ISNUM> in that the decimal character (as defined by decimalChar) is required to be in the string.

Example

```
<@ISDECIMAL STR="$2,345.00">
```

Evaluates to "1" (true)

```
<@ISDECIMAL STR="1.">
```

Evaluates to "1" (true)

```
<@ISDECIMAL STR="8495">
```

Evaluates to "0" (false)

See Also

<@ISALPHA>	page 174
<@ISALPHANUM>	page 175
<@ISINT>	page 181
<@ISNUM>	page 184
currencyChar	page 396
decimalChar	page 406
thousandsChar	page 465

<@ISINT>

Syntax

```
<@ISINT [STR=]string>
```

Description

Evaluates to "1" (true) if the string provided is a valid integer. An integer is a whole number, either negative or positive, including zero. If the string contains characters other than the digits (0-9) and the negative sign (-), this tag returns "0".

The evaluation of this tag differs from <@ISNUM> in that the currencyChar, decimalChar, and thousandsChar are not allowed.

Example

If <@ISINT> evaluates to "1" (true), then the value can be placed into a container that is typed as integer, such as SQL tinyint, smallint, int, and bigint columns.

```
<@ISINT STR="-3423">
```

Evaluates to "1" (true)

```
<@ISINT STR="$14.55">
```

Evaluates to "0" (false)

```
<@ISINT STR=<@CALC "172.43 + 64.99" precision=0>>
```

Evaluates to "1" (true) because a precision attribute of 0 removes the decimal place from the result of the <@CALC> tag thus returning an integer.

See Also

<@ISALPHA>	page 174
<@ISALPHANUM>	page 175
<@ISDECIMAL>	page 180
<@ISNUM>	page 184
currencyChar	page 396
decimalChar	page 406
thousandsChar	page 465

<@ISMETASTACKTRACE>

Syntax `<@ISMETASTACKTRACE>`

Description Evaluates to 1 if the Meta Stack Trace is available - that is, if the error occurred while processing the Results HTML (as opposed to when processing an action). In all other cases it will evaluate to 0.

See Also `<@METASTACKTRACE>` page 208

<@ISNULLOBJECT>

Syntax

```
<@ISNULLOBJECT OBJECT=variable [SCOPE=scope]>
```

Description

The <@ISNULLOBJECT> Meta Tag evaluates whether a variable is a null object. This tag returns 1 if the specified variable is an object variable *and* is null. The OBJECT attribute is required and specifies the name of an object variable. The SCOPE is optional, and specifies the scope of the object variable.

Example

If a TeraScript class file returned an object variable, you could use <@ISNULLOBJECT> to check whether this object was null before calling methods on it. Because the type of the returned variable from the TeraScript class file is "object", an object variable is always returned; however, if some error occurs within the TeraScript class file such that the particular object is not accessible, a null object is returned. This Meta Tag is useful for checking for such errors.

See Also

<@CALLMETHOD>	page 55
<@CREATEOBJECT>	page 83
<@NUMOBJECTS>	page 214
<@OBJECTAT>	page 216
<@OBJECTS></@OBJECTS>	page 217

<@ISNUM>

Syntax

<@ISNUM VALUE=*number*>

Description

Evaluates to non-zero if the expression specified in VALUE is a valid number. A number cannot contain characters other than numbers except the character(s) specified in currencyChar and the characters specified in decimalChar and thousandsChar to delimit parts of the string.

An empty or blank expression is not considered a valid number.

Examples

<@ISNUM VALUE="$1,000,000.00"> *true*

<@ISNUM VALUE="1 + 2"> *false*

See Also

currencyChar page 396
decimalChar page 406
<@ISDATE> page 176
<@ISTIME> page 176
<@ISTIMESTAMP> page 176
thousandsChar page 465

<@KEEP>

Syntax

<@KEEP STR=*string* CHARS=*char* [ENCODING=*encoding*]>

Description

Returns the string specified in STR stripped of all characters except those specified in CHARS. The operation of this Meta Tag is case sensitive. To retain both upper and lower case variations of a character include both characters in the CHARS.

Each of the attributes to <@KEEP> may include both literal values and Meta Tags that return strings.

Examples

```
<@KEEP STR="The quick fox" CHARS="aeiou">
```

This example evaluates to "euio".

```
<@KEEP STR="$200.00" CHARS="0123456789.">
```

This example evaluates to "200.00".

```
<@KEEP STR="This is the HTML" CHARS="TH">
```

This example evaluates to "THT".

```
<@KEEP STR="<COLUMN NAME=Invoice.totalcost>"
CHARS="0123456789.">
```

This example returns the value in the total cost column, stripped of any non-numeric characters.

See Also

Encoding Attribute page 8
<@OMIT> page 219

<@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. TeraScript 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 ldp 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 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' >
```

See Also

<@LDAPSEARCH> page 192
<@LDAPMODIFY> page 190
<@LDAPDELETE> page 188

<@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. TeraScript 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 ldp 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.	

Parameter	Required	Description	Default
TIMEOUT		Specifies a timeout for the query. This is how long to wait for a response from the server.	

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' >
```

See Also

<@LDAPSEARCH> page 192
<@LDAPMODIFY> page 190
<@LDAPADD> page 186

<@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. TeraScript 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 ldp 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' >
```

See Also

<@LDAPSEARCH> page 192
<@LDAPADD> page 186
<@LDAPDELETE> page 188

<@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. TeraScript 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 ldp 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 TeraScript 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">
```

See Also

<@LDAPADD> page 186
<@LDAPMODIFY> page 190
<@LDAPDELETE> page 188

<@LEFT>

Syntax

<@LEFT STR=*string* NUMCHARS=*numChars* [ENCODING=*encoding*]>

Description

Returns the first NUMCHARS characters from the string specified in STR and returns the extracted substring.

If the string contains any spaces—except for space embedded within Meta Tags—the string must be quoted.

Both STR and NUMCHARS attributes are mandatory. If a syntax error is encountered while the expression is parsed—no attributes at all, no string or no number of characters—the tag returns an empty string.

Examples

```
<@LEFT STR="alpha" NUMCHARS="3">
```

This example returns “alp”, the first three characters of “alpha”.

```
<@LEFT STR="<@INCLUDE
FILE='<@APPFILEPATH>BrownFox.txt'>" NUMCHARS="3">
```

This example returns “The”, the first three characters of “The Quick Brown Fox Jumps Over The Lazy Dog” (the contents of the BrownFox.txt file).

See Also

Encoding Attribute page 8
<@REPLACE> page 235
<@RIGHT> page 238
<@SUBSTRING> page 262

<@LENGTH>

Syntax

```
<@LENGTH STR=string>
```

Description

Returns the number of characters in the string specified in STR. The STR attribute may be a literal value, a Meta Tag that returns a value, or a combination of both.

Note For a more efficient means of returning the length of a variable, see <@VARINFO> <page $pagenum>.

Examples

```
<@LENGTH STR="This is a test">
```

This example evaluates to "14".

```
<@LENGTH STR="<@POSTARG NAME='SSN'>">
```

This example evaluates to the number of characters entered into the SSN form field.

```
<@LENGTH STR="<@COLUMN NAME='customer.lastname'>">
```

This example evaluates to the length of the customer's last name.

<@LITERAL>

Syntax

```
<@LITERAL VALUE=value [ENCODING=encoding]>
```

Description

Causes TeraScript to suppress Meta Tag substitution for the VALUE supplied.

One use for this Meta Tag is assigning Meta Tags to variables, as you need to do with the userKey, altUserkey, and domainScopeKey configuration variables.

Example

```
<@ASSIGN NAME="metaTag" VALUE="<@VAR NAME='myVar'>">
```

This would assign the value of the myVar variable to the metaTag variable, that is, not using the <@LITERAL> Meta Tag.

```
<@ASSIGN NAME="metaTag" VALUE="<@LITERAL
VALUE='<@VAR NAME="myVar">'">
```

This assigns the text "<@VAR NAME=myVar>" to the metaTag variable, using the <@LITERAL> Meta Tag.

See Also

domainScopeKey	page 412
Encoding Attribute	page 8
userKey, altuserKey	page 474

<@LOCATE>

Syntax

<@LOCATE STR=*string* FINDSTR=*substring*>

Description

Returns the starting position of *substring* in *string*. If *substring* is empty, omitted, or not in *string*, <@LOCATE> returns "0". If *substring* occurs more than once in *string*, the position of the first occurrence is returned.

The operation of <@LOCATE> is case sensitive. In order for a match to be found, *substring* must occur inside *string* exactly as it is specified, including case.

Each of the attributes to <@LOCATE> may be specified as a literal value, a Meta Tag that returns a value, or a combination of both.

Examples

```
<@LOCATE STR="A test string" FINDSTR="test">
```

This example evaluates to "3".

```
<@LOCATE STR="Not in here" FINDSTR="help">
```

This example evaluates to "0".

```
<@LOCATE STR="The rain in Spain" FINDSTR="ain">
```

This example evaluates to "6".

```
<@LOCATE STR="Welcome to my home page."
FINDSTR="come">
```

This example evaluates to "4".

```
<@LOCATE STR="TeraScript Studio"
FINDSTR="TeraScript">
```

This example evaluates to "0", because an exact match of "TeraScript", including case, is not found in the source string.

```
<@LOCATE STR="<@LOWER STR='TeraScript Studio'>"
FINDSTR="TeraScript">
```

This example evaluates to "1."

<@LOGMESSAGE>

Syntax

```
<@LOGMESSAGE MESSAGE=messagetext [LOGLEVEL=loglevel]
[TYPE={ACTIVITY*|EVENT}]>
```

Description

The <@LOGMESSAGE> Meta Tag allows you to print a message to the TeraScript Server log file.

Additionally, if debugging is on for the TeraScript application file or class file where the <@LOGMESSAGE> Meta Tag is specified, the message appears in the debug output if the LOGLEVEL attribute of the tag is set to 2 or greater.

The MESSAGE attribute specifies the text to be written to the log file.

The LOGLEVEL attribute is optional, and indicates the minimum log level at which the message is output. This value is compared to the current value of the configuration variable loggingLevel. This can be one of 1, 2, 3, 4, 5, or SUPPRESS; SUPPRESS means that no message is written to the log file. The default is 1.

The message appears in the log file or debug stream with the prefix [User Message].

Values for both attributes of this tag may contain Meta Tags.

Example

The following Meta Tags log any unauthorized attempts to access a certain area on a TeraScript-based Web site, and includes the name of the user attempting the access (using the userName variable):

```
<@LOGMESSAGE MESSAGE="Unauthorized access attempted
by user: <@VAR NAME='userName'>" LOGLEVEL="2">
```

See Also

loggingLevel page 435
logToResults page 436

<@LOWER>

Syntax

<@LOWER STR=*string* [ENCODING=*encoding*]>

Description

Returns the value specified in STR converted to lowercase. The STR attribute may be a literal value, a Meta Tag that returns a value, or a combination of both.

Examples

```
<@LOWER STR="This is a test">
```

This example evaluates to “this is a test”.

```
<@LOWER STR=<@POSTARG NAME=product_code>>
```

This example returns the contents of the form field product_code, converted to lowercase.

```
<@LOWER STR=<@COL NUM=1>>
```

This example returns the value from column one of the result set, converted to lowercase.

See Also

Encoding Attribute page 8
<@UPPER> page 280

<@LTRIM>

Syntax

<@LTRIM STR=*string* [ENCODING=*encoding*]>

Description

Returns the value specified in STR stripped of leading spaces. The STR attribute may be a literal value, a Meta Tag that returns a value, or a combination of both.

Examples

```
<@LTRIM STR="  this is padded  ">
```

This example returns "this is padded ".

```
<@LTRIM STR="<@COL NUM='2'>">
```

This example returns value of column 2, less any leading spaces.

See Also

Encoding Attribute	page 8
<@KEEP>	page 185
<@OMIT>	page 219
<@RTRIM>	page 241
stripCHARs	page 463
<@TRIM>	page 274

<@MAKEPATH>

Syntax

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

Description

The <@MAKEPATH> tag normalizes and combines paths to make a path of the requested type.

In its simplest form, when only one path is provided, the path is normalized - all path demlimiters 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 the second path will NOT be appended with the delimiter, so that a filename can be used) and combined into a single path, returned by the Meta Tag.

Note This tag may return ambiguous or unpredictable results when virtual directories within virtual hosts are used. Under these circumstances the tag should not be used.

Example

For non-Windows platforms:

```
<@MAKEPATH “c:\inetpub/wwwroot” “<@APPFILEPATH>”
TYPE=”FILESYSTEM”>
```

The above example will evaluate to:

```
c\:inetpub\wwwroot\appfilepath\
```

For Windows platforms:

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

The above example will evaluate to:

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

See Also

<@APPFILE> page 22
<@APPFILENAME> page 23
<@APPFILEPATH> page 24

<@MAP>

Syntax

```
<@MAP [NAME=]name [VALUE=]value [SCOPE=scope]
[ENCODING=encoding]>
```

Description

The <@MAP> Meta Tag takes an array and returns a single column array with the values of specified array cells of the same row, concatenated together. The VALUE attribute may contain Meta Tags that evaluate expressions (like <@IF>) which will use the value of each row to evaluate the expression.

Examples

Start with an array, which for this example contains age and gender, known as @@request$people:

```
<@ASSIGN request$people "<@ARRAY
value='Mr,Rawson,Cole,Male,2;Mrs,Joanne,Ball,Female
,40;,Madeline,Ryan,Saint,80;Mrs,Gemma,Falker,Female
,36;Ms,Briana,Fitzgerald,Female,8;Ms,Abbie,Savell,F
emale,10;'>">
```

Mr	Rawson	Cole	Male	2
Mrs	Joanne	Ball	Female	40
	Madelin e	Ryan	Saint	80
Mrs	Gemma	Falker	Female	36
Ms	Briana	Fitzgeral d	Female	7
Ms	Abbie	Savell	Female	10

To perform a simple <@MAP> concatenation on the people's names:

```
<@MAP request$people "#1 #2 #3" encoding="NONE">
```

Mr Rawson Cole
Mrs Joanne Ball
Madeline Ryan
Mrs Gemma Falker
Ms Briana Fitzgerald
Ms Abbie Savell

To perform an expression based concatenation which will give the females over 16 years of age flowers, the men beer, and, all others sweets:

```
<@MAP request$people value='<@IF "((#5 > 16) && (#4 =
Female))">Flowers: #1 #2 #3<@ELSEIF "#4 = Male">Beer:
#1 #2 #3<@ELSE>Sweets: #1 #2 #3</@IF>'>
```

Beer: Mr Rawson Cole
Flowers: Mrs Joanne Ball
Sweets: Madeline Ryan
Flowers:Mrs Gemma Falker
Sweets: Ms Briana Fitzgerald
Sweets: Ms Abbie Savell

To perform a conditional expression based <@MAP> concatenatation which will give the females over 16 years of age flowers, and, females under 16 years old sweets:

```
<@MAP request$people value='<@IF "((#5 > 16) && (#4
= Female))">Flowers: #1 #2 #3<@ELSEIF "((#5 <= 16)
&& (#4 = Female))">Sweets: #1 #2 #3</@IF>'>
```

Flowers: Mrs Joanne Ball
Flowers:Mrs Gemma Falker
Sweets: Ms Briana Fitzgerald
Sweets: Ms Abbie Savell

To perform an <@MAP> concatenation where the concatenation itself contains the expression which will give all the full names of the people with only the first letter of their first name

```
<@MAP request$people value='Full Name: #1 <@LEFT
STR="#2" NUMCHARS="1">. #3'>
```

Full Name : Mr R. Cole
Full Name : Mrs J. Ball
Full Name : M. Ryan
Full Name : Mrs G. Falker
Full Name : Ms B. Fitzgerald
Full Name : Ms A. Savell

See Also <@ARRAY> page 30

<@MAXROWS>

Description

Returns the value specified in the **Maximum Matches** field for the current Search or Direct DBMS action. If **No Maximum** was specified, <@MAXROWS> returns "0". This Meta Tag may be used only in a Search or Direct DBMS action.

This Meta Tag is especially useful when you specify a Meta Tag as the Maximum Matches value for a search action (allowing a form field or search argument value to determine, at execution time, the maximum number of matches to return).

Example

```
<@IFEQUAL VALUE1="<@MAXROWS>" VALUE2="0">
Here are the matching records:
<@ELSE>
Here are the <@MAXROWS> matching records:
</@IF>
<@ROWS>
...
</@ROWS>
```

This example indicates to you the maximum number of matches that are displayed.

See Also

<@NUMROWS> page 215
<@STARTROW> page 260
<@TOTALROWS> page 271

<@METAOBJECTHANDLERS>

Syntax

<@METAOBJECTHANDLERS [{*array attributes*}]>

Description

This Meta Tag returns an array with a row of three columns for each object-handling plug-in that were successfully loaded by the TeraScript Server on startup as well as static handlers (eg the TCF handler).

The first column of the returned array is the object handler's public name (for example, `TeraScript JavaBean Object Handler`); the second column is the type of object handler (a short string used internally by TeraScript to refer to the handler, such as `JavaBean`, `TCF`, or `COM`); the third column is the path to the handler (on Windows and Unix, this is the path to the library), which is used to identify the handler to the operating system when TeraScript wants to load it.

Row zero of the returned array contains the column headers.

Example

<@METAOBJECTHANDLERS> in a TeraScript application file returns:

On Windows:

TeraScript Class Files	TCF	[Internal]
COM / DCOM Objects	COM	TERASCRIPT_PATH\wshcm501.dll
Java Beans	JAVABEAN	TERASCRIPT_PATH\wshbn501.dll

On Linux:

TeraScript Class Files	TCF	[Internal]
Java Beans	JAVABEAN	TERASCRIPT_PATH/lshbn501.so

On OS X:

TeraScript Class Files	TCF	[Internal]
Java Beans	JAVABEAN	TERASCRIPT_PATH/mshbn501.so

See Also

Array-to-Text Attributes page 17

<@METASTACKTRACE>

Syntax `<@METASTACKTRACE>`

Description This Meta Tag returns a two dimensional array representing the Meta Stack Trace that occurs when an error occurs when processing Results HTML. The first column of the array is the line number on which the error occurred. The second column of the array is the Meta Tag that caused the error.

This tag will work with all the usual formatting attributes that will allow a user to change an array's appearance.

See Also `<@ISMETASTACKTRACE>` page 182

<@MIMEBOUNDARY>

Syntax

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

Description

This tag generates a MIME boundary string that can be used when composing mulitpart messages. If the parameter BOUNDARY is omitted (which is 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 levelID and the following alphanumeric sequence is the request scope identifier at the time when <@MIMEBOUNDARY> was being processed:

```
<@MIMEBOUNDARY LEVELID=1>
```

would return

```
----=MimePart__0001__33A8F4D74DE30EF93CBEFAA4
```

See Also

<@EMAIL> page 137
<@EMAILSESSION> page 140

<@NEXTVAL>

Syntax

<@NEXTVAL NAME=*variable* [SCOPE=*scope*] [STEP=*increment*]>

Description

Increments the specified variable by the specified increment and returns the new value. <@NEXTVAL> operates only on integer values. The default increment is "1", if no STEP is specified. You can specify a variable scope as well; see <@VAR> for a explanation of scoping rules.

If the variable does not exist, is non-integer, is not text, or if the step is non-integer, <@NEXTVAL> evaluates to nothing, and an error is logged if `LogLevel` is greater than 0.

Text variables (that is, standard variables) or individual array items may be updated by <@NEXTVAL>.

Example

Placing the following line in the Results HTML after each database access (Search, New Record, and so on) returns the number of times the user has accessed the database in their session:

```
<P>You have accessed the database
<STRONG><@NEXTVAL NAME="user$access"></STRONG>
times in this session.</P>
```

See Also

`loggingLevel` page 435
<@VAR> page 292

<@NULLOBJECT>

Description

The <@NULLOBJECT> Meta Tag allows you to create objects that do not do anything, but allow conditional tests that check whether a variable is empty to return a result. This tag can also be used to initialize object variables within the Assignment action.

Example

The following example assigns a variable myObject the value of an empty object whose value is undetermined.

```
<@ASSIGN NAME=myObject VALUE=<@NULLOBJECT>>
```

The <@IFEMPTY @@myObject> and <@ISNULLOBJECT @@myObject> Meta Tags return true and the LEN function of the <@CALC> Meta Tag yields a zero value. This Meta Tag could be used to return null objects when certain errors take place.

See Also

<@ASSIGN> page 33
<@CALC> page 43
<@CALLMETHOD> page 55
<@CREATEOBJECT> page 83
<@IFEMPTY> page 165
<@ISMETASTACKTRACE> page 182
<@NUMOBJECTS> page 214
<@OBJECTAT> page 216
<@OBJECTS></@OBJECTS> page 217

<@NUMAFFECTED>

Description

Returns the number of database rows affected by the last Insert, Update, Delete, or Direct DBMS action executed. All other actions have no effect on what the tag returns. The value returned by the tag is always the number of rows affected by the *last* Insert, Update, Delete, or Direct DBMS action executed. This tag only works for Oracle,
JDBC and ODBC data source types. The tag has no attributes.

At the start of execution, and until an Insert, Update, Delete, or Direct DBMS action is executed, the tag returns "-1".

Note

- If the last Direct DBMS action performed a search, the tag also returns "-1".
- Some ODBC drivers do not support this Meta Tag. For data sources using these drivers, the tag always returns "-1".

Example

An Update action in your application file updates a product code. In the Results HTML for that Update action, you could use <@NUMAFFECTED> to return to the user the number of records changed:

```
<P><STRONG><@NUMAFFECTED></STRONG> records were
updated in the database.</P>
```

<@NUMCOLS>

Syntax

```
<@NUMCOLS [ARRAY=array]>
```

Description

Returns the number of columns in each row.

Without the ARRAY attribute, this Meta Tag is valid in the Results HTML of any results returning action, and returns the number of columns in the result rowset.

With the optional ARRAY attribute, which accepts the name of a variable containing an array, the tag may be used anywhere Meta Tags are valid and returns the number of columns in the named array.

Example

```
Here are your results. There are <@NUMROWS> rows of
<@NUMCOLS> columns in the rowset

<@ROWS>
  <@COLS>
       <@COL>
  </@COLS>
<BR>
</@ROWS>
```

See Also

<@COLS> </@COLS>	page 75
<@CURCOL>	page 86
<@NUMROWS>	page 215

<@NUMOBJECTS>

Syntax

```
<@NUMOBJECTS OBJECT=objectvariable [SCOPE=scope]>
```

Description

This tag returns the count of the objects in a collection or iterator object returned by a COM or JavaBean method call.

The `OBJECT` attribute defines the name of a variable containing an object instance. This must be a collection or iterator. The optional `SCOPE` attribute defines the scope of the object variable.

Note For large collections, this Meta Tag could be very slow, as TeraScript must iterate through every item in order to get the count.

Example

The following `<@NUMOBJECTS>` could be used within an `<@OBJECTS>` loop:

```
Displaying 1 of <@NUMOBJECTS OBJECT=myCollection>
```

See Also

`<@CALLMETHOD>`	page 55
`<@CREATEOBJECT>`	page 83
`<@OBJECTAT>`	page 216
`<@OBJECTS></@OBJECTS>`	page 217

<@NUMROWS>

Syntax

```
<@NUMROWS [ARRAY=array]>
```

Description

Returns the number of rows in an action's result rowset or in the specified array.

Without the ARRAY attribute, this Meta Tag is valid in the Results HTML of any results returning action, and returns the number of rows in the result rowset.

With the optional ARRAY attribute, which accepts the name of a variable containing an array, the tag may be used anywhere that Meta Tags are valid, and returns the number of rows in the named array.

Example

```
<@NUMROWS> records were returned:<P>

<@ROWS>
<STRONG>Name:</STRONG> <@COLUMN
NAME="contact.name">
<BR>
<STRONG>Phone:</STRONG> <@COLUMN
NAME="contact.phone">
<BR>
</@ROWS>
```

This example returns a message indicating the number of records retrieved, then lists the name and phone number of each contact.

See Also

<@NUMCOLS> page 213
<@STARTROW> page 260
<@TOTALROWS> page 271

<@OBJECTAT>

Syntax

```
<@OBJECTAT OBJECT=variable NUM=index [SCOPE=scope]>
```

Description

Given an iterator or collection object (returned from a COM or JavaBean method call) and an index, this tag returns a single item from the object.

The `OBJECT` attribute defines the name of a variable containing an object instance. This must be a collection or iterator. The optional `SCOPE` attribute defines the scope of the object variable. The `NUM` attribute sets the index of the item to return (1 is the first item).

CautionThere is no direct access to collection items; the collection must be stepped through to reach a particular item. This can create poor performance in application files that use this tag.

The <@OBJECTAT> tag is not supported inside an <@OBJECTS> loop. Using it there may generate unpredictable results.

Example

The following assigns the first item in `myCollection` to `myItem`.

```
<@ASSIGN request$myItem VALUE=<@OBJECTAT
OBJECT=myCollection SCOPE=user NUM=1>>
```

Assuming that `myIterator` is a list of strings, the following example returns the third string.

```
<@OBJECTAT OBJECT=myIterator NUM=3>
```

See Also

<@CALLMETHOD>	page 55
<@GETPARAM>	page 154
<@CREATEOBJECT>	page 83
<@NUMOBJECTS>	page 214
<@OBJECTS></@OBJECTS>	page 217

<@OBJECTS></@OBJECTS>

Syntax

<@OBJECTS OBJECT=*objectvariable* ITEMVAR=*itemvariablename* [SCOPE=*objectscope*] [ITEMSCOPE=*itemvariablescope*] [START=*start*] [STOP=*stop*]></@OBJECTS>

Description

This Meta Tag loops through collection and iterator objects in variables returned by COM object and JavaBean method calls.

The OBJECT attribute defines the name of a variable containing an object instance. This must be a collection or iterator. The optional SCOPE attribute defines the scope of the object variable.

The ITEMVAR attribute defines the name of the variable in which to put the current item, and the optional ITEMSCOPE attribute defines the scope of the current item variable.

If the optional START attribute is specified, the loop skips the first (START-1) objects. If the attribute value is not a number, it is ignored. If the optional STOP attribute is specified, the loop stops after processing the item number given. If the attribute value is not a number, it is ignored.

Example

The following example loops through a collection object in request$foo that contains e-mail messages, and calls methods on each object within the collection to return Subject and Contents of the e-mail messages, and set the read flag:

```
<@OBJECTS OBJECT=foo SCOPE=request
ITEMVAR=request$currItem>Here is your unread
mail:<B>
<@IF EXPR="!(<@CALLMETHOD OBJECT=request$currItem
METHOD=ReadFlag() METHODTYPE=GET>)">
<@CALLMETHOD OBJECT=request$currItem
METHOD=Subject() METHODTYPE=GET></B><BR>
<@CALLMETHOD OBJECT=request$currItem
METHOD=Content() METHODTYPE=GET><BR>
<@CALLMETHOD OBJECT=request$currItem
METHOD=ReadFlag(1) METHODTYPE=SET>
<HR></@IF></@OBJECTS>
```

See Also

<@CALLMETHOD> page 55
<@GETPARAM> page 154
<@CREATEOBJECT> page 83

`<@NUMOBJECTS>` page 214
`<@OBJECTAT>` page 216

<@OMIT>

Syntax

<@OMIT STR=*string* CHARS=*char* [ENCODING=*encoding*]>

Description

Returns the value specified in STR stripped of all characters specified in CHARS. The operation of this Meta Tag is case sensitive. To omit both the upper and lower case variations of a character, you must include both characters in CHARS.

Each of the attributes of <@OMIT> may be specified using a literal value, Meta Tags that return values, or a combination of both.

Examples

```
<@OMIT STR="$200.00" CHARS="$">
```

This example evaluates to "200.00".

```
<@OMIT STR=" spacey" CHARS=" ">
```

This example evaluates to "spacey".

```
<@OMIT STR=green CHARS=gren>
```

This example evaluates to an empty string.

```
<@OMIT STR="$200.00" CHARS="01234567890.">
```

This example evaluates to "$".

```
<@OMIT STR="<@POSTARG NAME='PHONENUMBER'>"
CHARS="()-">
```

If the form field PHONENUMBER contains "(905) 819-1173" then this would evaluate to "9058191173".

```
<@OMIT STR="<@ARG NAME='user_input'>" CHARS=" <CHAR
9><@CRLF>">
```

This example returns the content of the argument named 'user_input' with white space characters (spaces, tabs, and returns) removed.

See Also

Encoding Attribute	page 8
<@KEEP>	page 185
<@LTRIM>	page 201
<@RTRIM>	page 241
<@TRIM>	page 274

<@PAD>

Syntax

```
<@PAD STR=string NUMCHARS=padToLength [CHAR=padcharacter]
[POSITION=BEFORE|AFTER] [ENCODING=encoding]>
```

Description

The <@PAD> Meta Tag returns the input string expanded to a specified length by prefixing or appending a given character as many times as necessary. It can be used to construct values to be passed to a function that expects fixed length data or to build up a table or other preformatted text for display in a monospaced font.

The STR attribute specifies the string to be padded.

The NUMCHARS attribute specifies the length to which the string is padded. If the specified string to be padded (STR) is longer than the length specified in NUMCHARS, the original string is returned.

The CHAR attribute specifies the character to use to pad the string. If more than one character is specified here, only the first character is used. If the CHAR attribute is absent, the space character is used to pad the string.

The POSITION attribute is optional, and indicates whether to pad the beginning (BEFORE) or end (AFTER) of the string. The default is AFTER.

All attributes of <@PAD> may contain Meta Tags.

Example

```
<@PAD STR="alpha" CHAR="x" NUMCHARS="8"
POSITION="after">
```

This example returns “alphaxxx”; that is “alpha” followed by three “x” characters for a total output length of 8 characters.

See Also

<@KEEP> page 185
<@LEFT> page 195
<@LENGTH> page 196
<@LTRIM> page 201
<@OMIT> page 219
<@RIGHT> page 238
<@RTRIM> page 241
<@SUBSTRING> page 262
<@TRIM> page 274

<@PLATFORM>

Syntax

```
<@PLATFORM [ENCODING=encoding]>
```

Description

Returns the platform name and bitness of the environment in which TeraScript Server is currently running. You may want to use this tag in Branch actions to branch to different External actions based on the current TeraScript Server platform. Currently, the <@PLATFORM> Meta Tag will always respond 32-bit, regardless of the bitness of the platform. This will be updated in a future release.

Example

<@PLATFORM> will evaluate to one of the following:

- Windows (32-bit)
- Mac OS X (32-bit)
- Linux (32-bit)

See Also

<@EDITION>	page 125
Encoding Attribute	page 8
<@PRODUCT>	page 224
<@VERSION>	page 302

<@POSTARG>

Syntax

```
<@POSTARG NAME=name [TYPE=type] [FORMAT=format]
[ENCODING=encoding]>
```

Description

Returns the value(s) of the named post argument (form field) in the HTTP request calling the application file. References to post arguments not present in the request evaluate to empty.

The NAME attribute may be specified as a literal value, value-returning Meta Tag, or a combination of both.

The TYPE attribute accepts one of two possible values: TEXT or ARRAY. ARRAY causes the tag to return a single-column, multi-row array of values, one for each value received for the named post argument. A <SELECT> form field with the MULTIPLE attribute, for example, sends multiple instances of the form field, one for each value selected by the user. Using the ARRAY type lets you access all those values. TEXT, which is the default type if the TYPE attribute is not specified, causes the tag to return a single value. If you specify this type when multiple values were received for the argument, the value returned is the first one received by TeraScript.

The optional FORMAT and ENCODING attributes determine how the value is formatted by TeraScript. These attributes are ignored if TYPE=ARRAY is specified.

Example

```
You asked for properties in <@POSTARG NAME="city">
```

This example includes the value from the form field "city" in the HTML.

See Also

<@ARG>	page 28
Encoding Attribute	page 8
Format Attribute	page 11
<@POSTARGNAMES>	page 223
<@SEARCHARG>	page 245
<@SEARCHARGNAMES>	page 246

<@POSTARGNAMES>

Syntax

```
<@POSTARGNAMES [{array attributes}]>
```

Description

Returns an array containing the names of all post arguments.

Post arguments are passed to TeraScript through forms. A form that has a method of POST returns the results of its fields through post arguments. <@POSTARGNAMES> provides a mechanism for identifying the names of all post arguments received in the current request.

The array returned has one column and *n* rows where there are *n* unique post arguments.

There are array-returning attributes that can be used to specify prefixes and suffixes for the returned array, rows within the array, and columns within the rows. They are described in the section Array-to-Text Attributes <page $pagenum>. By default, the returned array is formatted as an HTML table.

Example

The following returns all post argument names using the default array formatting:

```
<@ASSIGN NAME="mypostargs" VALUE="<@POSTARGNAMES>">
<@VAR NAME="mypostargs">
```

Note If multiple post arguments with the same name are received, the name of the post argument is listed only once.

See Also

Array-to-Text Attributes	page 17
<@ARG>	page 28
<@ARGNAMES>	page 29
<@SEARCHARGNAMES>	page 246

<@PRODUCT>

Syntax

<@PRODUCT [ENCODING=*encoding*]>

Description

Returns the name of TeraScript Server's product type.

Example

<@PRODUCT> on a licensed copy of TeraScript returns one of the following:

```
TeraScript Server
TeraScript Server (Trial)
```

See Also

<@EDITION> page 125
Encoding Attribute page 8
<@PLATFORM> page 221
<@VERSION> page 302

<@PURGE>

Syntax

<@PURGE [NAME=*name*] [SCOPE=*scope*]>

Description

Used to remove a variable from a scope, or to remove all variables from a scope.

Note The cookie scope, and the cookies known by a user's web browser, cannot be cleared with <@PURGE>. To make a cookie expire from a web browser, set its Expiry value to 'Now' in the Cookie Properties dialog of the Assign action, or set the EXPIRES attribute of the <@ASSIGN> Meta Tag to a GMT timestamp (encoded using datetime:http) in the past.

This Meta Tag cannot purge variables from the cookie and system scopes.

Examples

The following examples demonstrate how to remove a variable from a given scope:

```
<@PURGE NAME="foo" SCOPE="user">
<@PURGE NAME="foo" SCOPE="domain">
<@PURGE NAME="foo" SCOPE="request">
```

The following examples demonstrate how to remove all variables from a given scope:

```
<@PURGE SCOPE="user">
<@PURGE SCOPE="domain">
<@PURGE SCOPE="request">
```

See Also

<@ASSIGN> page 33
<@VAR> page 292

<@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 TeraScript's file cache. The design of this Meta Tag includes considerations for TeraScript 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 TeraScript 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.

Example

```
<@PURGECACHE>
```

Purges all files (tafs and includes) 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 includes) cached from all directories and all domains on the current TeraScript Server. Effectively, this clears 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 TeraScript Server then you will need to modify your existing <@PURGECACHE> code to include `domain="all"`.

See Also

`cacheIncludeFiles` page 387

<@PURGEDEBUG>

Description

Purges the accumulated debug output up to the point that the tag is executed.

Note For best results, this tag should be used alone in a Result action, or placed at the very top of the desired action results.

See Also

<@PURGERESULTS> page 229

<@RESULTS> page 237

<@PURGERESULTS>

Description Clears the Results HTML accumulated in previous actions.

Note <@PURGERESULTS> does not clear the accumulated results of the current action. For best results, this tag should be used alone in a Result action, or placed at the very top of the desired action results.

See Also <@RESULTS> page 237

<@RANDOM>

Syntax

<@RANDOM [HIGH=*high*] [LOW=*low*]>

Description

Returns a random number between HIGH and LOW, inclusive of their values.

The HIGH and LOW attributes may range from zero to 2,147,483,647. If only one attribute is specified, a number between zero and that number is returned. If no attribute is specified, a number between zero and 32767 is returned.

Either of the attributes for <@RANDOM> may be specified using literal values or by using Meta Tags that return values.

Examples

```
<@RANDOM HIGH="100" LOW="1">
```

This example returns a random number between 1 and 100.

```
<@RANDOM LOW="1" HIGH="<@NUMROWS>">
```

This example returns a random number between 1 and the number of rows returned by the current action.

```
<@RANDOM HIGH="<@POSTARG NAME='pickANumber'>">
```

This example returns a random number between zero and the `pickANumber` form field value submitted with the current request.

See Also

<@CALC> page 43

<@REGEX>

Syntax

<@REGEX EXPR=*expression* STR=*text* TYPE=*type*>

Description

Provides an interface to POSIX regular expression matching routines from inside TeraScript. This gives you powerful tools to match text patterns if they are needed.

<@REGEX> accepts as attributes the regular expression (EXPR), the text to match the pattern against (STR), and the type of the regular expression (TYPE), basic or extended. If the attributes contain spaces, they must be quoted—single or double, as appropriate. <@REGEX> returns its results in the form of an array and should be assigned to a variable via <@ASSIGN>.

Upon a successful match, <@REGEX> returns an array with three columns and *n*+1 rows, where *n* is the number of parenthesized subexpressions in the pattern. The first column contains the matching text, the second column contains the start index of the matching portion, and the third column gives the length of the matching portion. The start and length are compatible with the <@SUBSTRING> tag.

Rows *i* from 1 to *n* give the *i*th matching parenthesized subexpression, and row *n*+1 gives the entire matching portion of the text. (If there are no parenthesized subexpressions, the whole match is returned in the first row.)

The table gives a sample array returned from <@REGEX>.

<@REGEX EXPR="([[:alpha:]]+),[[:space:]]+([A-Z]{2})[[:space:]]+([A-Z][0-9][A-Z] [0-9][A-Z][0-9])" STR="in Mississauga, ON L5N 6J5." TYPE=E>.

Mississauga	4	11
ON	17	2
L5N 6J5	20	7
Mississauga, ON L5N 6J5	4	23

In the above example, <@REGEX> is using 3 sub-query expressions to form an overall regular expression:

1. Any number of alpha followed by a comma and any number of spaces thereafter;

2 then 2 upper case letters followed by any number of spaces;

3 then an upper case letter followed by a number followed by a lower case letter, then a space, followed by a number, followed by an upper case letter, followed by a number.

All 3 must be satisfied for a match.

If attributes are missing, <@REGEX> returns a string with the problem attributes. Upon an error condition, <@REGEX> returns a single character, "C" for a pattern compile failure, and an "M" for a match failure. If any attributes are missing, a textual message is displayed indicating the missing items. You can easily test for success by using <@VARINFO NAME=*variable* ATTRIBUTE=TYPE>.

Tip For more information on constructing POSIX regular expressions, ask your local UNIX guru, consult the FreeBSD regex man page, or try doing an Internet search for the term "POSIX 1003.2".

<@RELOADCONFIG>

Description

This Meta Tag forces a reload of the following configuration files:

- TeraScript Server Configuration File
- `terascript.ini`
- Object Configuration File
- Application Configuration File
- Domain Configuration File
- Timed URL processing setup file.

TeraScript Server writes out all changed configuration variable values to the TeraScript Server configuration file before reloading.

Security Feature

This tag requires that a user scope `configPasswd` variable with the same value as the system `configPasswd` exists when it is executed; otherwise, an error is generated and the configuration files are not reloaded.

<@RELOADCUSTOMTAGS>

Syntax

```
<@RELOADCUSTOMTAGS [SCOPE=system|application]>
```

Description

This Meta Tag forces a reload of the custom tags files of the specified scope. For more information on custom tags, see `Custom Meta Tags` `<page $pagenum>`.

The default value of the `SCOPE` attribute is `SYSTEM`.

Security Feature

This tag requires the `configPasswd` for the scope requested. If a user scope `configPasswd` variable with the same value as the system or application scope `configPasswd` does not exist, an error is generated and the tag file is not reloaded.

This Meta Tag does not return a value.

See Also

`<@CUSTOMTAGS>` page 90

<@REPLACE>

Syntax

<@REPLACE STR=*string* FINDSTR=*findString* REPLACESTR=*replaceString* [POSITION=*position*][TYPE=*regex*] [ENCODING=*encoding*]>

Description

Returns a text string in which all the occurrences of FINDSTR in the value specified in STR are replaced with the substitute as specified in REPLACESTR. If the POSITION attribute is specified, only that occurrence of FINDSTR is replaced.

Strings that contain spaces must be quoted.

If a syntax error is encountered while the expression is parsed—no attributes at all, no string, no keyword, no substitute, or no occurrence—the tag returns an empty string.

Regular expressions can be used in the FINDSTR attribute. This allows for more complex find and replace functions to be constructed. The optional parameter TYPE=Regex is used to flags the FINDSTR as a regular expression.

<@REPLACE> is case insensitive.

Examples

```
<@REPLACE STR="alpha" FINDSTR="a" REPLACESTR="u"
POSITION="2">
```

This example returns "alphu", replacing the second occurrence of "a" with "u".

```
<@REPLACE STR="<@INCLUDE
FILE='<@APPFILEPATH>BrownFox.txt'>"
FINDSTR="<@INCLUDE
FILE='<@APPFILEPATH>BrownFox.txt'>" REPLACESTR="A">
```

This example replaces "The Quick Brown Fox Jumps Over A Lazy Dog" (the content of the BrownFox.txt file) with "A".

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

would produce the following string:

```
allballball
```

See Also

Encoding Attribute	page 8
<@LEFT>	page 195
<@LOCATE>	page 198

<@REGEX>	page 231
<@REPLACE>	page 235
<@RIGHT>	page 238
<@SUBSTRING>	page 262

<@RESULTS>

Syntax

```
<@RESULTS [ENCODING=encoding]>
```

Description

Evaluates to the accumulated Results HTML for the current execution of the application file.

The returned value includes the Results HTML for all the actions up to, but not including, the current action.

The accumulated Results HTML can be cleared with the <@PURGERESULTS> tag.

Example

This tag can be used to give a variable the value of the Results HTML from a database query so that the results can be used in other application file calls without re-doing the search. (This technique is useful only with data that does not change often—a list of product categories, for example.) After generating the HTML and assigning <@RESULTS> to a variable (cached_list, for example), subsequent calls to the application file can be handled by checking the contents of the variable with a Branch action. If cached_list is not empty, you can immediately return <@VAR NAME="cached_list">. If the variable is empty, you would branch to the normal processing to query the database.

See Also

Encoding Attribute page 8
<@PURGERESULTS> page 229

<@RIGHT>

Syntax

<@RIGHT STR=*string* NUMCHARS=*numChars* [ENCODING=*encoding*]>

Description

Extracts the last number of characters from the string specified in STR and returns the extracted substring.

If the string contains any spaces—except for spaces embedded within Meta Tags—it must be quoted.

Examples

```
<@RIGHT STR="alpha" NUMCHARS="3">
```

This example returns "pha", the last three characters of "alpha", beginning from the right.

```
<@RIGHT STR="<@INCLUDE
FILE='<@APPFILEPATH>BrownFox.txt'>" NUMCHARS="3">
```

This example returns "Dog", the last three characters of "The Quick Brown Fox Jumps Over The Lazy Dog" (the content of the BrownFox.txt file).

See Also

Encoding Attribute	page 8
<@LEFT>	page 195
<@LOCATE>	page 198
<@REGEX>	page 231
<@REPLACE>	page 235
<@SUBSTRING>	page 262

<@ROWS> </@ROWS>

Syntax

```
<@ROWS [ARRAY=array] [SCOPE=scope] [START=start]
[STOP=stop] [STEP=step]></@ROWS>
```

Description

The Results HTML appearing between this tag pair is processed once for each row of the result set generated by an action.

This tag pair also allows iteration over the rows of an array. This tag places a copy of the text between the opening and closing tags for each row of the array.

ARRAY is the array to loop over. It can be the name of an array variable or an array value. The default value is `resultSet`. All results-returning actions (Search, Direct DBMS, External, Script, and Mail) perform an automatic assignment of their results array to the request variable `resultSet`.

START refers to the starting value for the index. The default value is 1.

STOP refers to the stopping value for the index. The loop terminates when this value is exceeded, not when it is reached. The default value is <@NUMROWS>.

STEP refers to the increment added to the index after each iteration. The default value is 1.

PUSH, a deprecated attribute [PUSH=*push*], allows the sending of data. This attribute has been deprecated as of the release of TeraScript 6. It is currently operational but will be removed in the next major version release. Developers are encouraged to discontinue use of this Meta Tag. When applicable, a warning will be reported to the `witangoevents.log` file.

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

Note This tag must appear in pairs and cannot span multiple actions. START and STOP can only be used to specify points inside the array. If the index exceeds the number of rows in the result set or reaches a negative value, the loop terminates. If the specified STEP does not take the index from START to STOP, no iterations are made. If the START equals the STOP, one iteration is made, regardless of the step or array sizes.

<@ROWS> blocks can be nested. In that case, the tags that get their reference from a <@ROWS> block (for example, <@COL>, <@COLUMN>, <@MAXROWS>) refer to the innermost <@ROWS> block.

Examples

```
<@ROWS ARRAY="<@VARNAMES SCOPE='USER'>"
START="<@MAXROWS>" STOP="1" STEP="1">
Variable <@CURROW> is named <@COL NUM="1"> <BR>
</@ROWS>
```

Variable *x* is named *varname*. It is printed for each variable in the user's scope, going in reverse order.

```
<@ROWS PUSH=100>
       <@COLUMN NAME="ACTIVITYLOG.LOGTIMESTAMP">
       <@COLUMN NAME="ACTIVITYLOG.DOMAINNAMEID"><BR>
</@ROWS>
```

This example allows you to see the resulting HTML 100 rows at a time. The effects of the PUSH attribute depend on the HTML presentation of the result set and the Web browser that is used to access TeraScript. Sometimes, even though TeraScript and the Web server are sending data to the Web browser, the Web browser holds up the data without displaying it. For example, if the <@ROWS> block in the previous paragraph sits between a <TABLE></TABLE> with rows of the result set corresponding to the rows of the table, a Netscape Web browser does not display the result file until the HTML <TABLE> block is completed.

See Also

<@COL> page 74
<@COLUMN> page 76

<@RTRIM>

Syntax

<@RTRIM STR=*string* [ENCODING=*encoding*]>

Description

Returns the value specified in STR stripped of trailing spaces. The STR attribute may be a literal value or a Meta Tag that returns a value.

This Meta Tag is useful for stripping spaces from the end of CHAR column values returned from DBMSs such as Oracle, which pad values to the declared length of the column. You may also use the stripChars configuration variable to accomplish this task.

Example

```
<@RTRIM STR=" this is padded ">
```

This example returns " this is padded"

```
<@RTRIM STR="<@COL NUM='2'>">
```

This example returns value for column two, less any trailing spaces.

See Also

Encoding Attribute	page 8
<@KEEP>	page 185
<@LTRIM>	page 201
<@OMIT>	page 219
stripCHARs	page 463
<@TRIM>	page 274

<@SCRIPT>

Syntax

<@SCRIPT [SCOPE=*scope*]>*script here*</@SCRIPT>

or

<@SCRIPT EXPR=*expr* [SCOPE=*scope*]>

Description

Used for server-side execution of scripts written in JavaScript.

The tag syntax can take one of two forms, and which one you use depends on how much script you have. Functionally, the two forms are equivalent and the result of evaluating the tag is the output from the script; so, for example, <@SCRIPT EXPR="2+2"> evaluates to "4".

Usage One: <@SCRIPT [SCOPE=scope]> your script here</@SCRIPT>

This is the block form of the tag. You can use this syntax for large chunks of script where it makes sense for the script to be blocked out by begin/end tags. The script can contain other TeraScript tags; those tags are substituted *prior* to script execution. In order for the script to be able to interact with the TeraScript environment, there are predefined object/methods that can be called from the script. For more information, see Predefined Objects <page $pagenum>.

The optional SCOPE attribute defines the lifetime of the objects and functions declared in the script, and is similar to the scope of variables. All TeraScript scopes are supported. The default scope is REQUEST, so anything defined in one script can be referenced in another script in the same file execution. IMMED, which is specific to script executions, specifies that the execution context for the script is completely deleted immediately after running the script, and is used to ensure no name/space clashes occur between the script and other longer-lived objects. You can also specify METHOD, INSTANCE, USER, APPLICATION, and DOMAIN scopes, or a custom scope.

Nesting of <@SCRIPT> blocks is not supported.

Usage Two: <@SCRIPT EXPR="your script here"

[SCOPE=scope]>

This is a shorthand form of the tag for small script snippets. As with the long form of this tag, the script snippet can contain other TeraScript tags that are substituted *prior* to script execution.

Note If the script expression attribute is supplied, then it is syntactically invalid to include the closing </@SCRIPT> tag, and the closing tag is left unsubstituted. Also note that all attributes to <@SCRIPT> must be named.

Predefined Objects

The following predefined objects and methods exist in the JavaScript environment of TeraScript Server to allow scripts to interact with TeraScript in a controlled and meaningful way:

- **server**: object representing TeraScript Server.
- **getVariable(name)**: gets TeraScript a variable. Using default scoping rules, returns variable value.
- **getVariable(name, scope)**: as in the previous paragraph, but defined with scope.
- **setVariable(name, value)**: sets a TeraScript variable, using default scoping rules, returns nothing.
- **setVariable(name, value, scope)**: as in the previous paragraph, but with defined scope.

Note TeraScript variables are accessed by value, not by reference. You must therefore use *setVariable* to update TeraScript with any changes you make to variable values. Also, because they are passed by value, getting large TeraScript arrays can consume a lot of memory because the entire array is duplicated inside of JavaScript.

Because TeraScript supports only two-dimensional arrays, it is an error to try to put a JavaScript array of more than two dimensions into a TeraScript variable.

For more information on the JavaScript capabilities of TeraScript, see the online help for JavaScript that is distributed with TeraScript (in the `Help` directory under the TeraScript root directory).

Examples

```
<@SCRIPT EXPR="1*2*3*4">
```

This example returns a value of “24”.

```
<@SCRIPT EXPR="server.setVariable ('foo', 'bar');">
```

This example sets the TeraScript variable “foo” to the value “bar”, so that a subsequent `<@VAR NAME="foo">` returns “bar”.

```
<@SCRIPT EXPR="server.getVariable('foo');">
```

This example is equivalent to `<@VAR NAME="foo">`.

See Also

`<@ASSIGN>`	page 33
`<@VAR>`	page 292

<@SEARCHARG>

Syntax

```
<@SEARCHARG NAME=name [TYPE=type] [FORMAT=format]
[ENCODING=encoding]>
```

Description

Returns the value(s) of the named search argument (name/value pairs after a "?" in the URL, or form fields in a GET method form) in the HTTP request calling the application file. References to search arguments not present in the request evaluate to empty.

The NAME attribute may be specified as a literal value, value-returning Meta Tag, or a combination of both.

The TYPE attribute accepts one of two possible values: TEXT or ARRAY. ARRAY causes the tag to return a single-column, multi-row array of values, one for each value received for the named search argument. A URL like `http://www.yoursite.com/my.taf?x=1&x=2&x=3`, for example, sends three separate values for the x search argument. Using the ARRAY type lets you access all those values. TEXT, which is the default type if the TYPE attribute is not specified, causes the tag to return a single value. If you specify this type when multiple values were received for the argument, the value returned is the first one received by TeraScript.

The optional FORMAT and ENCODING attributes determine how the value is formatted by TeraScript. These attributes are ignored if TYPE=ARRAY is specified.

Example

```
The items in the <@SEARCHARG NAME="category_name">
category are:

<@ROWS>
<@COLUMN NAME="product.name"><BR>
</@ROWS>
```

This example includes the requested category name in a heading prior to listing the products.

See Also

<@ARG>	page 28
Encoding Attribute	page 8
Format Attribute	page 11
<@POSTARG>	page 222

<@SEARCHARGNAMES>

Syntax

<@SEARCHARGNAMES [{*array attributes*}]>

Description

Returns an array containing the names of all search arguments.

Search arguments are passed to TeraScript through the URL.

For the URL:

```
http://hostname/path_to_cgi/
path_to_taf?sarg1=value1&sarg2=value2&...
&sargn=val
```

<@SEARCHARGNAMES> returns an array containing a subset of the names `sarg1, sarg2,...,sargn`. The result array has one column and *n* rows where there are *n* unique search arguments.

There are array-returning attributes that can be used to specify prefixes and suffixes for the returned array, rows within the array, and columns within the rows. They are described in the section `Array-to-Text Attributes <page $pagenum>`. By default, the returned array is formatted as an HTML table.

Example

The following returns all search argument names using the default array formatting:

```
<@SEARCHARGNAMES>
```

See Also

Array-to-Text Attributes	page 17
<@ARG>	page 28
<@ARGNAMES>	page 29
<@POSTARGNAMES>	page 223

<@SECSTODATE>, <@SECSTOTIME>, <@SECSTOTS>

See the following Meta Tags:

<@DATETOSECS>, <@SECSTODATE>page 95
<@TIMETOSECS>, <@SECSTOTIME>page 266
<@TSTOSECS>, <@SECSTOTS> page 275

<@SERVERNAME>

Description

This Meta Tag returns the name of the TeraScript Server that is processing the current application file. The name is determined by the stanza name for the TeraScript Server in the TeraScript Server configuration file (`terascript.ini`). This tag can be used while using multiple TeraScript Servers to share load (load splitting).

<@SERVERNAME> has no attributes.

Example

```
<@SERVERNAME>
```

Returns the name of the TeraScript Server you are running. An example of a returned TeraScript Server name is:

```
TeraScript__Server
```

<@SERVERSTATUS>

Syntax

```
<@SERVERSTATUS [VALUE=value] [ENCODING=encoding]
[{array attributes}]>
```

Description

Returns status information on TeraScript Server. The tag has an optional attribute, `VALUE`. The value of this attribute must be one of the categories specified in the following table (case insensitive).

If the value attribute is not specified, a two-column array is returned, giving all status values with the category name in the first column and the value in the second column. With this form of the tag, the `ENCODING` attribute, if specified, is ignored.

There are array-returning attributes that can be used to specify prefixes and suffixes for the returned array, rows within the array, and columns within the rows. They are described in the section `Array-to-Text Attributes <page $pagenum>`. By default, the returned array is formatted as an HTML table.

Category	Description
ActiveDataSrc	number of data sources marked in use.
ActiveQryThr	number of threads marked in use.
AvgAFReadTime	average amount of time taken to read an application file from disk, cache, or network, in milliseconds, accurate to 1/60 second.
AvgAFSize	average size of application file read from disk, cache, or network.
AvgQryProcTime	average time to process a request, in milliseconds, accurate to 1/60 second.
AvgQryReadTime	average time to read a prepare a request for processing, in milliseconds, accurate to 1/60 second.
AvgQryWriteTime	average time to return results to the user after processing, in milliseconds, accurate to 1/60 second.
CacheBytesUsed	number of bytes used by application and include files in TeraScript Server's file cache. If caching is turned off, this category returns "0".
DataSrcCount	number of data source connections allocated.
HeapSize	current amount of working memory consumed (bytes).

Category	Description
LstQryProcTime	time to process last request, in milliseconds, accurate to 1/60 second.
MinQryProcTime	minimum request processing time so far, in milliseconds, accurate to 1/60 second.
MaxQryProcTime	maximum request processing time so far, in milliseconds, accurate to 1/60 second.
MaxQryThreadCount	maximum number of threads that are in use at one time in the server.
NumAFRead	number of TeraScript application files read from disk, cache, or network.
NumCachedDocs	number of documents in the application file cache.
NumCachedIncl	number of documents in the include file cache.
NumQryErrors	number of requests ended by an error since TeraScript Server's inception.
NumQrykilled	number of requests killed (timed out) since server inception.
NumQryServed	number of requests served since server inception.
NumTripBadConn	number of network CGI-server connections failed.
NumUsersLocal	number of user references in the request variable store.
NumUsersShared	number of user references in the shared variable store.
NumVarsLocal	number of variables in the request variable store.
NumVarsShared	number of variables in the shared variable store.
ProcessID	system process ID number.
ProcessSize	current size of server process (bytes).
QryLapsedTime	time elapsed since the last query was executed, in milliseconds since the restart of the current operating system — used by the Server Watcher to check whether TeraScript Server is running correctly.
QryStartTime	time at which a query thread was started, in milliseconds since the restart of the current operating system — used by the Server Watcher to check whether TeraScript Server is running correctly.
QryThrCount	number of processing threads allocated.
TotlTripRdBytes	total number of bytes read via network.
TotlTripRdFiles	total number of files read via network.

Category	Description
TotlTripRdTime	total amount of time taken to read all files read via network, in milliseconds, accurate to 1/60 second.
UpTime	server running time (in minutes).
VariableStoreSize	number of bytes used by all variables in all shared scopes and the local scope of the currently processed request.
Version	version of status information (not the server). TeraScript returns "62".

Notes


- `NumAFRead`, `AvgAFReadTime`, and `AvgAFSize` include cache reads, and reflect the performance of the application file cache.
- `QryProc` values may include time taken to push intermediate data back to the user.
- `NumQryServed` includes any requests killed and tallied in `NumQryKilled`.
- *Windows and Linux only:* `TripRd` values measure actual network and disk accesses, and represent true overhead to read include/application files that are new, uncached, or changed since being cached.

Retrieving these values via the tag means executing a request, which affects the status values as specified:

- A thread is used to process the request, and that bumps up the `ActiveQryThr` count.
- Since the current request has not completed yet, the `QryProcTime` times and the `NumQryServed` count do not reflect the currently executing request.
- Executing a request involves reading it, so `NumAFRead` and the `AvgAF` values reflect the current request.
- If a network read was required to load the application file and other include files, the `TotlTripRd` values are updated.

See Also

`Encoding Attribute` page 8

<@SETCOOKIES>

Description

For use in an HTTP header. Returns the correct `Set-Cookie` lines to set the values of cookie variables assigned in the current application file execution.

Note Make sure you include this Meta Tag in any custom headers you create for TeraScript. If you do not, the cookie scope does not work properly.

See Also

`headerFile` page 424
`<@HTTPATTRIBUTE>` page 156
`<@HTTPSTATUSCODE>` page 160

<@SETPARAM>

Syntax

<@SETPARAM NAME=*name* VALUE=*value*>

Description

<@SETPARAM> sets the value of a parameter variable within a TeraScript class file. This tag is similar to <@ASSIGN>, but performs error checking to ensure that only Out and In/Out parameters of a TeraScript class file can be set.

This Meta Tag is specifically used for setting the value of a parameter in a TeraScript class file. If the variable specified by the NAME attribute is not a TeraScript class file In/Out or Out parameter, this tag returns an error.

This tag is only valid within a TeraScript class file method.

The NAME attribute specifies the name of the parameter to assign the value to. Similar restrictions apply to parameter names assigned by <@SETPARAM> as apply to all variables; see <@ASSIGN> <page $pagenum>.

Note Because the parameter variables specified by <@SETPARAM> are only valid in method scope, scope cannot be specified in the NAME attribute, unlike the <@ASSIGN> Meta Tag (for example, NAME=request$foo generates incorrect results).

The VALUE attribute specifies the value to assign to the variable. The VALUE attribute may specify text, an array (using the <@ARRAY> Meta Tag or an array variable), a DOM variable, or an object variable.

Arrays

If the parameter being assigned to exists and contains an array, this tag also lets you set the values of individual elements in that array. <@SETPARAM> can assign an array (or array section) to a variable, or to another array (or array section). Array assignments require that the source and target arrays (or array sections) have the same dimensions.

If you are assigning to an array variable element or section, the name includes the element or section specification specified within square brackets as [rownumber,colnumber], with an asterisk indicating all rows or all columns; for example, NAME=myArray[1,2] or NAME=myArray[*,3].

If you are assigning to an array section, the value specified here must match the dimensions of the array variable specification in NAME.

Example

Within the Results HTML of a TeraScript class file method, you could use the following series of Meta Tags to get the value of an In parameter (in this case, the radius of a sphere), perform calculations on it (calculating the surface area of a sphere), and set the value of a returned (Out) parameter accurate to two decimal places:

```
<@SETPARAM NAME=OutSurface VALUE=<@CALC
EXPR="4*P*(<@GETPARAM NAME=Radius>^2)"
PRECISION=2>>
```

See Also

<@ASSIGN> page 33
<@GETPARAM> page 154

<@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.

<@SORT>

Syntax

```
<@SORT ARRAY=arrayVarName [COLS=sortCol [sortType]
[sortDir] [, ...]] [SCOPE=scope]>
```

Description

Sorts the input array by the column(s) specified. This tag does not return anything.

The ARRAY attribute specifies the name of a variable containing an array. The COLS attribute specifies the column(s) to sort by, specified using column numbers or names, with optional sort types (*sortType*) and directions (*sortDir*).

Valid sort types are SMART (the default), DICT, ALPHA and NUM. DICT sorts the column alphabetically, irrespective of case. ALPHA is a case-sensitive sort. NUM sorts the column numerically. SMART checks whether values are numeric or alphabetic and sorts using a NUM or DICT type.

Valid sort directions are ASC (the default) and DESC. ASC sorts the column in ascending order, with lower values coming before higher ones. DESC sorts in descending order, with higher values coming before lower ones.

If the COLS attribute is omitted, all columns are sorted left to right using the SMART sort type and the ASC (ascending) sort direction.

The order of the type and direction options are not important, that is, `COLS="1 NUM ASC"` is equivalent to `COLS="1 ASC NUM"`.

Multiple columns may be specified, separated by commas. Each sort column specification may include a sort type specifier and/or a sort direction specifier. If included, these must follow the sort column, separated by a space.

Multiple sort columns cause the array to be sorted by the first column specified, then, rows with the same value in that column are sorted by the second sort column specified within that previously-created sort order, and so on.

The SCOPE attribute specifies the scope of the variable specified by ARRAY. If not specified, the default scoping rules are used.

Meta Tags are permitted in any of the attributes.

Examples

- If the request variable `test` contains the following array:

4	example
2	is
7	sorting
3	an
5	of
1	here
6	array

The same array in sorted order can be gotten by using `<@SORT ARRAY="test" SCOPE="request" COLS="1 NUM">@@request$test,` as shown in the following example:

1	here
2	is
3	an
4	example
5	of
6	array
7	sorting

- `<@SORT ARRAY="customer" COLS="cust_state, cust_num">` sorts the array stored in `customer`. The default `SMART` sort type checks the `cust_state` column, finds it is alphabetic, and uses sort type `DICT`; similarly, it checks the `cust_num` column, finds it is numeric, and uses sort type `NUM` in the `cust_num` column for the rows with the same `cust_state` value.

See Also

`<@DISTINCT>` page 105
`<@FILTER>` page 148
`<@INTERSECT>` page 171
`<@UNION>` page 277

<@SQ>

Description

To use single and double quotes inside a Meta Tag attribute value, use <@SQ> for a single quote "'" and <@DQ> for a double quote "".

Example

```
<@ASSIGN NAME="Important_Quote" VALUE="Yoda said,
<@DQ>Do, or do not; there is no
<@SQ>try<@SQ>.<@DQ>">

<@VAR NAME="Important_Quote">
```

This example returns the following:

```
Yoda said, "Do, or do not; there is no 'try'."
```

See Also

<@DQ>, <@SQ> page 120

<@SQL>

Syntax `<@SQL [ENCODING=encoding]>`

Description Returns last action-generated SQL.
This Meta Tag accesses your last action-generated SQL.

See Also `Encoding Attribute` page 8

<@STARTROW>

Description

Returns the position of the first row retrieved by a Search or Direct DBMS action within the set of records matching the action's criteria. This value corresponds to the one specified in the **Start retrieval at match number** field in the Results section of the Search action.

Example

```
<@TOTALROWS> records matched your criteria. Listed
here are <@NUMROWS> records, starting with record
<@STARTROW>.

<@ROWS>
...
</@ROWS>
```

This example returns a message indicating the number of records found and returned, and the position of the first record shown within the found rowset.

See Also

<@ABSROW>	page 18
<@CURROW>	page 89
<@NUMROWS>	page 215
<@ROWS> </@ROWS>	page 239
<@TOTALROWS>	page 271

<@STOP>

Syntax <@STOP>

Description Immediately ceases execution of the request.

This tag operates as if an error has occurred, breaking out of loops and nested elements and completing request processing at that point, without actually raising an error. Accumulated results will be returned to user unless @PURGERESULTS is executed first. One side effect of this tag is that NumQryErrors reported by @SERVERSTATUS will be incremented.

<@SUBSTRING>

Syntax

<@SUBSTRING STR=*str* START=*start* NUMCHARS=*numChars* [ENCODING=*encoding*]>

Description

Extracts a NUMCHARS long substring, starting at START from STR and returns a copy of the extracted substring.

If the string contains any spaces except for spaces embedded within Meta Tags, the string must be quoted.

All three attributes are mandatory. If a syntax error is encountered while the expression is parsed (no attributes at all, no string, or no number of characters) the tag returns an empty string.

Examples

```
<@SUBSTRING STR="alpha" START="3" NUMCHARS="2">
```

This example returns "ph", the two characters starting at the third position.

```
<@SUBSTRING STR="<@INCLUDE
FILE='<@APPFILEPATH>BrownFox.txt'>" START="3"
NUMCHARS="2">
```

This example returns "e ", that is, an E and a space, which are the two characters starting at the third position in "The Quick Brown Fox Jumps Over The Lazy Dog" (the contents of the BrownFox.txt file).

See Also

Encoding Attribute	page 8
<@LEFT>	page 195
<@LOCATE>	page 198
<@REPLACE>	page 235
<@RIGHT>	page 238

<@THROWERROR>

Syntax

```
<@THROWERROR [NUMBER=string] [DESCRIPTION=string]>
```

Description

This Meta Tag generates ("throws") an error with the specified number and description.

Errors generated with this Meta Tag cause the same behavior as built-in TeraScript errors:

- If the current action has Error HTML, the Error HTML is processed.
- If the current action has no Error HTML, the system-wide default error message is processed (defined by the `defaultErrorFile` configuration variable; by default, this is `error.htx`).
- If there is no system-wide default error message, TeraScript's built-in error message is used.

Execution then halts and the processed error message is returned.

Tip See <@CLEARERRORS> <page $pagenum> for a way to continue execution of an application file after an error occurs.

The ability to generate your own errors is useful when handling error conditions or other exception cases in your applications. Instead of handling these cases in your main code, you can use <@THROWERROR> and put the error-handling parts of your code in Error HTML. This makes for cleaner, more readable, and more maintainable code.

The optional NUMBER attribute defines the number of the error that is generated. If the attribute is not present, empty, or is not a number, `0` is returned.

Tip All built-in TeraScript errors codes are negative (for example, -3,
-108). It is recommended that you use positive error codes so they do not conflict with current or future built-in errors.

The optional DESCRIPTION attribute contains a text description of the error.

You can access the values for the error number and the error description from the Error HTML or the system-wide default error message, using the <@ERROR> Meta Tag with the PART="number1" and PART="message1" attributes, respectively. The class of all errors generated by the <@THROWERROR> Meta Tag is "Application". Use <@ERROR PART="class"> in Error HTML to access the class of an error.

This Meta Tag cannot be used in either Error HTML or in the system-wide default error message.

Example

```
<@IF EXPR="<@ARG thePassword>='Correct Password'">
      Welcome Back!
<@ELSE>
      <@THROWERROR NUMBER="88334" DESCRIPTION="You
have entered the wrong password.">
</@IF>
```

If linked with a user's logging in and if the wrong password is entered, this example displays the built-in TeraScript error (assuming there is no system-wide default error message and no Error HTML associated with the action), which resolves to the following:

```
Error

An error occurred while processing your request:
File: myfile.taf
Position: Login_Check
Class: Application

Main Error Number: 88334

You have entered the wrong password.
```

See Also

<@CLEARERRORS> page 73
<@EMAIL> page 137
<@ERRORS> </@ERRORS> page 145

<@TIMER>

Syntax

<@TIMER [NAME=*name*] [VALUE=*value*]>

Description

Allows you to create and use named timers. These timers exist only within the scope of a single application file execution. <@TIMER> accepts and returns its numbers in milliseconds.

Upon application file start-up, the default timer named ELAPSED is created to track elapsed time, and is set to zero.

You can create new timers or update existing ones by calling <@TIMER> with an optional NAME and a required VALUE attribute. If the name attribute is not specified, the default timer (ELAPSED) is updated with the value specified.

If a non-numeric value is given, the timer is set to zero. Values may be negative. When setting a timer, the tag returns nothing.

The value of a timer is retrieved if only the NAME attribute and no VALUE attribute is specified. Retrieving a non-existent timer returns nothing.

Because NAME is optional, the most simple and direct use of the tag is <@TIMER>, which returns the elapsed time for the current application file.

Examples

```
<@TIMER>
```

Returns the value of the default timer (ELAPSED).

```
<@TIMER VALUE="-30000">
```

Sets the value of the default timer (ELAPSED) to -30,000 milliseconds.

```
<@TIMER NAME="Fred" VALUE="3000">
```

Creates a new timer named Fred and sets its value to 3000 milliseconds.

```
<@TIMER NAME="Fred">
```

Returns the current value of Fred.

<@TIMETOSECS>, <@SECSTOTIME>

Syntax

```
<@TIMETOSECS TIME=time [FORMAT=format]>
<@SECSTOTIME SECS=seconds [FORMAT=format]
[ENCODING=encoding]>
```

Description

<@TIMETOSECS> checks the entered time and, if valid, converts it into seconds. Conversely, <@SECSTOTIME> converts the entered seconds to a time.

For details, see <@ISDATE>, <@ISTIME>, <@ISTIMESTAMP> <page $pagenum>.

Both handle ODBC, ISO, and some numeric formats.

If the time is entered incorrectly—wrong separators or a nonexistent number of hours, minutes or seconds—the tag returns, "Invalid time!".

The TIME attribute is mandatory. If no attribute is found while the expression is parsed, the tag returns "No attribute!".

Examples

```
<@TIMETOSECS TIME=02:00:04>
```

This example returns "7204", the number of seconds contained in two hours and four seconds.

```
<@SECSTOTIME SECS=7204>
```

This example returns "02:00:04", the time in hour, minute and second format, assuming that is how times are configured with the timeFormat configuration variable.

See Also

Encoding Attribute	page 8
<@FORMAT>	page 153
Format Attribute	page 11
<@ISDATE>	page 176
<@ISTIME>	page 176
<@ISTIMESTAMP>	page 176
timeFormat	page 468

<@TMPFILENAME>

Syntax

```
<@TMPFILENAME [ENCODING=encoding]>
```

Description

Generates a unique temporary file name on the file system that TeraScript Server is currently executing on.

Example

```
<@ASSIGN NAME="myfile1" VALUE="<@TMPFILENAME>">
<@ASSIGN NAME="myfile2" VALUE="<@TMPFILENAME>">
```

The `myfile1` and `myfile2` variables can now be referenced in a File action (for example, writing interim information to a temporary scratch file) and are guaranteed to be unique file names on the system running TeraScript Server.

See Also

`Encoding Attribute` page 8

<@TOGMT>

Syntax

```
<@TOGMT TS=timestamp [ENCODING=encoding] [FORMAT=format]>
```

Description

Transforms local time, given by the TS argument, to GMT (Greenwich Mean Time). The transformed time can be formatted according to the optional FORMAT attribute.

The difference between GMT and local time is influenced by daylight savings time. That is, for New York, New York, the regular difference is 5 hours, but the summertime difference is 4 hours. TeraScript accounts for daylight savings time.

<@TOGMT> is a synonym of <@TOUTC>.

Note When a two-digit year is given, the following centuries are assumed:

Value	Century
00-36	2000s
37-99	1900s

For example, a two-digit year of 99 is evaluated as 1999, and a two-digit year of 00 is evaluated as 2000.

Example

```
<@TOGMT TS="<@CURRENTTIMESTAMP>">
```

See Also

<@CURRENTTIMESTAMP> page 88
Encoding Attribute page 8
Format Attribute page 11

<@TOKENIZE>

Syntax

```
<@TOKENIZE STR=text CHARS=delimiters  [NULLTOKENS={TRUE | FALSE}] [{array attributes}] [CDELIM=columnDelimString] [RDELIM=rowDelimString]>
```

Description

Provides you with a way of sectioning a string into multiple pieces according to a set of delimiting characters. It accepts as attributes the STR of the text and the delimiting CHARS, and returns its results as an array. The result is a either a one row array, with a column for each token, or, a two dimensional array where CDELIM and RDELIM are specified. If the entire string consisted of only delimiters, a one by one empty array is returned.

Note In prior versions of TeraScript the attribute STR was called VALUE. As this was an inconsistency in the meta language the attribute name was changed. To ensure backward compatibility, the attribute name VALUE will continue to be supported.

Each character in CHARS is taken as a separate delimiter.

There are array-returning attributes that can be used to specify prefixes and suffixes for the returned array, rows within the array, and columns within the rows. They are described in the section Array-to-Text Attributes <page $pagenum>. By default, the returned array is formatted as an HTML table.

The **NULLTOKENS** attribute can be used with this tag to recognize empty tokens. It may be set to *true* to process empty tokens, or, *false* to skip them. If NULLTOKENS is ommitted, the behaviour assumes the value of false.

Example

Note There are extra spaces in the string specified in the VALUE attribute in the following example.

```
<@TOKENIZE VALUE="  There is no ‘try’. Do, or do not. " CHARS=" ,.">
```

The array returned looks like this:

There	is	no	‘try’	Do	or	do	not

The following tokenize action

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

would return the two dimensional array shown below:

this is text	and more
and more	

See Also

Array-to-Text Attributes page 17

<@TOTALROWS>

Description

Returns the total number of rows matching the criteria specified in the most recently executed Search action.

This tag returns a meaningful value only if the **Get total number of matches** option is selected in the Results section of the Search action.

The value is set to -1 at the start of execution and whenever a Select action does not have the total number of rows option set. In all other cases the tag will evaluate to the total rows of the most recently executed Select.

Example

```
<@TOTALROWS> records matched your criteria. Listed
here are <@NUMROWS> records, starting with record
<@STARTROW>.

<@ROWS>
...
</@ROWS>
```

This example returns a message indicating the number of records found and the number shown.

See Also

<@ABSROW>	page 18
<@CURROW>	page 89
<@NUMROWS>	page 215
<@ROWS> </@ROWS>	page 239
<@STARTROW>	page 260

<@TOUTC>

Syntax

```
<@TOUTC TS=timestamp [ENCODING=encoding] [FORMAT=format]>
```

Description

Transforms local time, given by the TS argument, to UTC (Coordinated Universal Time). The transformed time can be formatted according to the optional FORMAT attribute.

The difference between UTC and local time is influenced by daylight savings time. That is, for New York, New York, the regular difference is 5 hours, but the summertime difference is 4 hours. TeraScript accounts for daylight savings time.

<@TOUTC> is a synonym of <@TOGMT>.

Note When a two-digit year is given, the following centuries are assumed:

Value	Century
00-36	2000s
37-99	1900s

For example, a two-digit year of 99 is evaluated as 1999, and a two-digit year of 00 is evaluated as 2000.

Example

```
<@TOUTC TS="<@CURRENTTIMESTAMP>">
```

See Also

<@CURRENTTIMESTAMP> page 88
Encoding Attribute page 8
Format Attribute page 11
<@TOGMT> page 268

<@TRANSPOSE>

Syntax

<@TRANSPOSE ARRAY=*arrayVarName* [SCOPE=*scope*] [{*array attributes*}]>

Description

Exchanges row and column specifications for values in an array; for example, the value in the third row, first column is transposed to the first row, third column. The ARRAY attribute specifies the array to transpose. The optional SCOPE attribute specifies the scope.

This tag returns a new array. The original array is not modified; you can use the <@ASSIGN> Meta Tag or Assign action to assign the result of this tag to a variable.

There are array-returning attributes that can be used to specify prefixes and suffixes for the returned array, rows within the array, and columns within the rows. They are described in the section Array-to-Text Attributes <page $pagenum>. By default, the returned array is formatted as an HTML table.

Example

If the request variable fred contains a 3 x 4 array of the following form:

1	USA	1.72
2	Canada	84.2
3	Brazil	34
4	Argentina	47

```
<@ASSIGN NAME="fred_transposed" VALUE='<@TRANSPOSE
ARRAY="fred" SCOPE="request">'>
```

@@fred_transposed returns a 4 x 3 array of the following form:

1	2	3	4
USA	Canada	Brazil	Argentina
1.72	84.2	34	47

See Also

<@ASSIGN> page 33

<@TRIM>

Syntax

<@TRIM STR=*string* [ENCODING=*encoding*]>

Description

Returns the value specified in STR stripped of leading and trailing spaces. The value of the STR attribute may be a literal value or a Meta Tag that returns a value.

Examples

```
<@TRIM STR="  this is padded  ">
```

This example returns "this is padded".

```
<@TRIM STR="<@COL NUM='2'>">
```

This example returns the value of column "2", less any leading and trailing spaces.

See Also

Encoding Attribute	page 8
<@KEEP>	page 185
<@LTRIM>	page 201
<@OMIT>	page 219
<@RTRIM>	page 241

<@TSTOSECS>, <@SECSTOTS>

Syntax

<@TSTOSECS TS=*timestamp* [FORMAT=*format*]>
<@SECSTOTS SECS=*seconds* [FORMAT=*format*] [ENCODING=*encoding*]>

Description

<@TSTOSECS> checks the entered timestamp and converts it into seconds, using as a reference, midnight (00:00:00) January 1, 1970 (1970-01-01). Conversely, <@SECSTOTS> converts the entered seconds to a timestamp.

These tags support dates in the range 1970–2037.

All formats assume the Gregorian calendar. All years must be greater than zero. <@TSTOSECS> handles ODBC, ISO, and some numeric formats.

If the date is entered incorrectly—wrong separators or a nonexistent number of hours, minutes or seconds—the tag returns "`Invalid day!`" If the time is entered incorrectly (wrong separators or a nonexistent number of hours, minutes or seconds) the tag returns "`Invalid time!`".

The time attribute is mandatory. If no attribute is found while the expression is parsed, the tag returns "`No attribute!`".

If the optional `FORMAT` attribute is not used, the value in the configuration variable `timestampFormat` is used as the output format of this tag.

Examples

```
<@TSTOSECS TS="2000-01-01 12:00:00">
```

This example returns "946728000".

```
<@SECSTOTS SECS="5">
```

This example returns "1970-01-01 00:00:05", assuming the default configuration variables correspond to the example's format.

See Also

<@DATETOSECS>	page 95
Encoding Attribute	page 8
<@FORMAT>	page 153
Format Attribute	page 11
<@ISDATE>	page 176
<@ISTIME>	page 176
<@ISTIMESTAMP>	page 176

`<@SECSTODATE>` page 95
`<@SECSTOTIME>` page 266
`timestampFormat` page 401
`<@TIMETOSECS>` page 266

<@UNION>

Syntax

```
<@UNION ARRAY1=arrayVarName1 ARRAY2=arrayVarName2
[COLS=compCol [compType]] [SCOPE1=scope1]
[SCOPE2=scope2]>
```

Description

Returns the union of two arrays. The union consists of the combination of both arrays, with duplicates removed. Duplicates are found based on the values of the specified columns, checked using the specified comparison type.

The two input arrays are not modified. To store the result of this Meta Tag in a variable, use a variable assignment.

Note To join two arrays without removing duplicates, use the <@ADDROWS> tag.

The ARRAY1 and ARRAY2 attributes specify the names of variables containing arrays. The optional COLS attribute specifies the column(s) to consider when eliminating duplicates: the columns are specified using column numbers or names, with an optional comparison type (*compType*). The arrays must have the same number of columns; otherwise, an error is generated.

Valid comparison types are SMART (the default), DICT, ALPHA and NUM. DICT compares columns alphabetically, irrespective of case. ALPHA performs a case-sensitive comparison. NUM compares columns numerically. SMART checks whether values are numeric or alphabetic and performs a NUM or DICT comparison.

If no COLS attribute is specified, the elimination of duplicates is accomplished via a SMART comparison type that examines all columns in a row.

The SCOPE1 and SCOPE2 attributes specify the scope of the variables specified by ARRAY1 and ARRAY2, respectively. If the attribute is not specified, the default scoping rules are used.

Meta Tags are permitted in any of the attributes.

Examples

- If the variable old_items contains the following array:

blue
green

orange

and the array `new_items` contains the following:

orange
pink
blue
pink

`<@UNION ARRAY1="old_items" ARRAY2="new_items">` returns:

blue
green
orange
pink

- If the variable `test` contains the following array:

1	a	a
2	b	c
3	c	c

and the variable `test2` contains:

1	a	a
2	b	b
3	c	c
3.0	c	c

`<@UNION ARRAY1="test" ARRAY2="test2">` returns:

1	a	a
2	b	c
3	c	c
2	b	b

- Variable `usr1` contains the following:

Gilbert	Steve	1823-1344	$433.00
Brown	Robert	5543-1233	$332.50
Brown	Marsha	1122-5778	$541.00

Variable usr2 contains the following:

Kelly	Herbert	5543-1443	$100.50
Brown	Robert	6670-1123	$1123.75
MacDonald	Bill	1551-0787	$150.75

To find the unique users in both arrays, you would find the union of the two arrays based on the first two columns.

```
<@UNION ARRAY1="usr1" ARRAY2="usr2" COLS="1, 2">
```

returns:

Gilbert	Steve	1823-1344	$433.00	
Brown	Robert	6670-1123	$1123.75	*
Brown	Marsha	1122-5778	$541.00	
Kelly	Herbert	5543-1443	$100.50	
MacDonald	Bill	1551-0787	$150.75	

* TeraScript returns just one of the rows that have the same values in the specified columns (1 and 2).

The values in columns 3 and 4 are ignored for the purpose of the union operation since COLS="1, 2" is specified.

See Also

<@ADDROWS> page 20
<@DISTINCT> page 105
<@FILTER> page 148
<@INTERSECT> page 171
<@SORT> page 256

<@UPPER>

Syntax

<@UPPER STR=*string* [ENCODING=*encoding*]>

Description

Returns the string specified in STR converted to uppercase. The value of the STR attribute may be a literal value or a Meta Tag that returns a value.

Examples

```
<@UPPER STR="This is a Test">
```

This example returns "THIS IS A TEST".

```
<@UPPER STR="<@POSTARG NAME='product_code'>">
```

This example returns the contents of the form field `product_code`, converted to uppercase.

See Also

`Encoding Attribute` page 8
<@LOGMESSAGE> page 199

<@URL>

Syntax

```
<@URL LOCATION=location [BASE=base] [USERAGENT=useragent]
[FROM=from] [ENCODING=encoding] [USERNAME=username]
[PASSWORD=password] [POSTARGS=postarglist]
[POSTARGARRAY=arrayvariable] [WAITFORRESULT=true|false]
[DETAILEDRESPONSE=true|false] [MAXRESULTSIZE=size]>
```

Description

Requests the specified URL and returns its data, stripped of the HTTP header.

<@URL> supports HTTP URLs, and HTTPS URLs. The HTTP-type URL must be of the following form:

```
http[s]://hostname:port/path?search-arguments
```

where *port* defaults to 80 if not specified, and *path* and *search-arguments* are defaulted to an empty list.

The BASE attribute adds the specified value as an HTML <BASE> tag (that is, <BASE HREF=base>) within the HTML <HEAD> element of the retrieved HTML. This is necessary to load any inline data (for example, images) that are specified in relative URL format on the page retrieved. TeraScript prepends the specified value of the BASE attribute to the relative path.

The USERAGENT attribute is placed in the User-Agent line of the request. If USERAGENT is not specified, or is empty, the value of the userAgent configuration variable is used. For more information, see "userAgent" in the *Meta Tags and Configuration Variables* manual.

The User-Agent value in HTTP requests gives the destination server information about the program (such as, name, version, and platform) that is requesting the URL. For example, the User-Agent value passed by Netscape Navigator 4.04 for Windows NT is:

```
Mozilla/4.04 [en] (WinNT; I)
```

Servers often use the user agent information to determine the format of the results returned. (TeraScript application files can get the user agent information from a request using <@HTTPATTRIBUTE NAME="USER_AGENT">.) For example, a server may return a special version of a page, including Web browser-specific HTML for additional features, when the Web browser is Netscape Navigator or Microsoft Internet Explorer.

Use the USERAGENT attribute when you want TeraScript Server to emulate a specific Web browser so the server returns the data in the format you want.

The FROM attribute sets the value for the From line of the HTTP header specified for in the <@URL> Meta Tag.

You should use the FROM attribute to specify the e-mail address of the person to contact if the URL request is causing problems at the destination server. Supplying an e-mail address is especially important when the <@URL> Meta Tag is included in an application file that is executed automatically using TeraScript's timed URL processing functionality. If something goes wrong, the destination server administrator knows who to contact.

(TeraScript application files can get the FROM information from a request using <@HTTPATTRIBUTE NAME="FROM_USER">.)

If you do not specify a value, the default value is given by the configuration variable, mailDefaultFrom.

The optional USERNAME attribute is used to indicate the username required to communicate with a protected site.

The optional PASSWORD attribute is used to indicate the password required to communicate with a protected site.

The username and optional password can also be specified using the standard URL syntax:

```
<@URL LOCATION=
"http://username:password@www.example.com/">
```

If this syntax is used, it overrides username and password values specified using the USERNAME and PASSWORD attributes.

The optional POSTARGS attribute specifies the post content for the request, for example, a list of name-value pairs. They may not be specified with an array variable; to specify post arguments with an array, use the POSTARGARRAY attribute.

The names and values must be separated with = (equal sign) characters, and name-value pairs must be separated with & (ampersand) characters. Additionally, the names and values must be encoded. You may perform this encoding using the <@URLENCODE> Meta Tag: TeraScript does not automatically encode data passed in the POSTARGS attribute.

The optional POSTARGARRAY attribute is used to specify post arguments (name-value pairs) with an array. The value of POSTARGARRAY is the name of a variable containing an array of exactly two columns: the first column of the array must contain the names, and the second column must contain the values. TeraScript

extracts these from the array and uses them in the HTTP request. If an array of more than two columns is referenced, an error is returned.

When the POSTARGS or POSTARGARRAY attribute is present, the type of HTTP request issued by the <@URL> Meta Tag changes to POST from GET.

Sometimes, you use the <@URL> tag to trigger processing of a task but you do not care about the result. In this situation, performance of a TeraScript application is improved if TeraScript Server does not have to wait for the result of the HTTP request. The optional WAITFORRESULT attribute is used to indicate whether TeraScript Server waits for the results of the HTTP request. Possible values are true and false. The default is true. If the WAITFORRESULT attribute is set to false, <@URL> does not return a value.

The optional attribute MAXRESULTSIZE allows a size limitation to be placed on the results received from the <@URL> target server. The default value for this attribute is 64K, the minimum value may vary but can be as small as 512 bytes.

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

The optional DETAILEDRESPONSE attribute is used to indicate the type of response returned by <@URL>. Possible values are true and false. The default is true. If this attribute is set to false, <@URL> returns the HTML content resulting from the HTTP request; if this attribute is set to true, <@URL> returns an XML document containing the information resulting from the HTTP request.The format of this document is shown in the example below.

Detailed Response Format

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE HTTP_RESPONSE>
<HTTP_RESPONSE>
<STATUS>
       <CODE>200</CODE>
       <TEXT>OK</TEXT>
</STATUS>
<HEADER NAME="Date"><![CDATA[Day, Date, Time]]>
</HEADER>
<HEADER NAME="Server"><![CDATA[AV/1.0.1]]></HEADER>
```

```
<HEADER NAME="MIME-Version"><![CDATA[1.0]]></HEADER>
<HEADER NAME="Content-Length"><![CDATA[18576]]>
</HEADER>
<HEADER NAME="Content-Type"><![CDATA[text/html]]>
</HEADER>
<HEADER NAME="Set Cookie"><![CDATA[Sample Cookie]]>
<HEADER>
<BODY><![CDATA[<html>[page goes here]</html>]]></BODY>
</HTTP_RESPONSE>
```

XML Element	Description
<HTTP_RESPONSE>	The content of the HTTP parsed into an XML format.
<STATUS>	Contains the <CODE> and <TEXT> fields.
<CODE>	The HTTP code returned by the URL request.
<TEXT>	The returned HTTP response status message.
<HEADER>	The NAME attribute of <HEADER> describes the type of HTTP parameter returned. The <HEADER> element contains the content of the HTTP parameter.
<BODY>	The page's HTML in a CDATA block.

The <@URL> Meta Tag returns an error message if a time out or error condition occurs.

Note If you intend to display the value of <@URL> in a Web browser when DETAILEDRESPONSE is set to `false`, you must use the ENCODING=NONE attribute.

Windows only

The Windows TeraScript Server supports 40-bit SSL (Secure Sockets Layer) connections for use in encryption. On Windows 2000/NT with 128-bit SSL support, TeraScript Server supports 128-bit SSL connections.

Note TeraScript Server on Unix platforms uses the open SSL library. HTTPS calls may be affected by different versions on this library.

Examples

```
<@URL LOCATION="http://www.example.com/">
```

Returns the front page of `http://www.example.com/`.

```
<@URL LOCATION="http://www.example.com/"
BASE="http://www.example.com/"
USERAGENT="Mozilla/4.04 [en] (WinNT; I)"
FROM="TeraScript-admin@mycompany.com"
ENCODING="NONE">
```

Returns the front page of `http://www.example.com/` with a defined user agent, base URL, and from attribute.

```
<@URL LOCATION="https://auscommerce1.TeraScript.com">
```

This uses HTTPS to connect you to TeraScript's online store system via SSL.

See Also

`Encoding Attribute`	page 8
`mailDefaultFrom`	page 438
`userAgent`	page 473

<@URLDECODE>

Syntax

<@URLDECODE STR=*string*>

Description

This Meta Tag decodes strings encoded in URL format, such as strings encoded with <@URLENCODE> Meta Tag or passed in the HTTP header; for example, the value of <@HTTPATTRIBUTE NAME=HTTP_COOKIE>.

The STR attribute specifies the string to be URL-decoded. It may be specified using text, a variable, or another Meta Tag.

Example

```
<@URLDECODE STR="Hello%20World">
```

This example returns "Hello World".

See Also

<@URL> page 281
<@URLENCODE> page 287

<@URLENCODE>

Syntax

<@URLENCODE STR=*string*>

Description

Makes this string specified in STR compatible for inclusion in a URL by escaping characters that have special meaning in URLs, such as spaces and slashes according to the protocol specified in RFC 1630.

This tag works exactly like the ENCODING=URL attribute, but can be used to URL-encode any value.

Examples

```
<@URLENCODE STR="Hello World">
```

This example returns "Hello%20World".

```
<@URLENCODE STR="<@ACTIONRESULT NAME='action1'
NUM='1'>">
```

This example returns the result of the <@ACTIONRESULT> with special characters escaped.

See Also

<@URL> page 281

<@USERREFERENCE>

Description

Returns a unique number identifying the user executing the application file in which the tag appears. If no user reference number was received (via the "_userReference" search argument or an HTTP cookie) when the application file was called, a new number is generated; otherwise, the number passed in is returned.

The user reference number can be used for reliable tracking of user variables.

See Also

`userKey, altuserKey`	page 474
`<@USERREFERENCEARGUMENT>`	page 289
`<@USERREFERENCECOOKIE>`	page 290

<@USERREFERENCEARGUMENT>

Description

Evaluates to _userReference=<@USERREFERENCE>.

This Meta Tag is intended for use in Results HTML anchor URLs when you are tracking user variables by user reference and require support for Web browsers that do not support HTTP cookies.

Example

```
<A HREF="<@CGI>/shop/
add_item_to_basket.taf?item=29&
<@USERREFERENCEARGUMENT>">Add item</A>
```

This example includes the user's user reference ID value in the URL of the link.

See Also

userKey, altuserKey	page 474
<@USERREFERENCE>	page 288
<@USERREFERENCECOOKIE>	page 290

<@USERREFERENCECOOKIE>

Description

Used in the default HTTP header of TeraScript when returning results. It permits intelligent setting of the user reference cookie, a value that can be used to track user variables.

If no TeraScript user reference number was received, either via cookie or search argument, with the current HTTP request, <@USERREFERENCECOOKIE> returns the following:

```
Set-Cookie: Witango_UserReference=<@USERREFERENCE>;
path=/[CRLF]
```

([CRLF] stands for a carriage return/linefeed (ASCII 13/10) combination.)

If a user reference number was received with the current HTTP request, <@USERREFERENCECOOKIE> returns nothing. Because the cookie has already been set, there is no need to set it again.

Example

This is the content of TeraScript's default HTTP header::

```
HTTP/1.1 200 OK [CRLF]
Server: <name of webserver> [CRLF]
MIME-Version: 1.0 [CRLF]
Content-type: text/html [CRLF]
<@USERREFERENCECOOKIE>[CRLF]
```

See Also

userKey, altuserKey	page 474
<@USERREFERENCE>	page 288
<@USERREFERENCEARGUMENT>	page 289

<@UUID>

Description

Outputs a Universally Unique Identifier based on RFC 4122 version 4.

Version 4 UUIDs use a scheme relying only on random numbers. This algorithm sets the version number as well as two reserved bits. All other bits are set using a random or pseudorandom data source. Version 4 UUIDs have the form `xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx` where `x` is any hexadecimal digit and `y` is one of `8`, `9`, `A`, or `B`.

Example

`f47ac10b-58cc-4372-a567-0e02b2c3d479`.

<@VAR>

Syntax

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

Description

<@VAR> retrieves the contents of a variable, and, depending on the operation being performed, formats the data appropriately. Any of the attribute values of <@VAR> may be specified by other Meta Tags.

For more information on variables see Working With Variables page 319

Text

When retrieving the contents of a text (standard variable), the result of <@VAR> is always a text string.

Arrays

<@VAR> may also be used to retrieve an array. However, <@VAR> does different things to arrays based on context: <@VAR> converts the array to text whenever the result of the tag is returned in Results HTML, or when TYPE=text is specified; <@VAR> returns an internal reference to the array when it is used to copy an array from one place to another. So, if <@VAR> is used within <@ASSIGN>, then no conversion to text is performed (unless the TYPE="text" attribute is specified).

There are array-returning attributes that can be used to specify prefixes and suffixes for the returned array, rows within the array, and columns within the rows. They are described in the section Array-to-Text Attributes <page $pagenum>. By default, the returned array is formatted as an HTML table.

DOM (XML document instance)

<@VAR> can be used to retrieve all or part of a document instance (XML) variable. The XPATH and XPOINTER parameters can be used access to elements in DOM variables.

A document instance is returned by <@VAR> as XML with no conversion of characters to HTML entities if the ENCODING attribute is not present. No conversion occurs even when XML is placed in HTML (for example, to be displayed as a Web page). All other ENCODING attribute settings function normally.

You can display encoded XML by first assigning the XML text to a variable (using the TYPE=TEXT attribute, which forces XML to be returned, not a document instance), and then returning the value of that variable in the HTML. The default encoding of variables returned with <@VAR> then takes place, for example:

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

The XML is returned with the appropriate text converted to HTML entities.

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.

Scoping Rules

Scoping is the method by which variables can be organized and disposed of in an orderly and convenient fashion. It is highly recommended that all variables be explicitly scoped when referenced. There are various levels of scoping, each of which has an appropriate purpose:

For more information, see "Configuration Variables" on page 373.

- **System Scope** contains any variables that are general to all users. This scope contains only TeraScript Server configuration variables. To use this scope, specify SCOPE=system or SCOPE=sys.

For more information, see "domainScopeKey" on page 412 .

- **Domain Scope** contains variables that users can share if they are accessing a particular TeraScript application file from a specified TeraScript domain. TeraScript domains are specified in a domain configuration file, or default to the domain name (base URL or IP address) of the path to the TeraScript application file. This scope is defined by setting the system configuration variable domainScopeKey appropriately; that is, setting it to a value that can differentiate such users. By default, this is <@DOMAIN>, which returns the value of the current TeraScript domain.To use this scope, specify SCOPE=domain.
- **Application Scope** contains variables that are shared across TeraScript applications. TeraScript applications are defined by TeraScript users in an application configuration file. To use this scope, specify SCOPE=application or SCOPE=app.
- **User Scope** contains variables that a user defines and expects to be able to access from many application files or invocations of single application files. To use this scope, specify SCOPE=user or SCOPE=usr.

- **Request Scope** contains variables that should be unique to every invocation of any application file. For example, this scope could be used for temporary variables that reformat output from a search action. All variables of this scope are removed when the application file concludes execution. To use this scope, specify `SCOPE=request`, or `SCOPE=doc`.
- **Instance Scope** contains variables that are valid in an instance of a TeraScript class file. These variables can be shared across methods called on a TeraScript class file, if the methods are called on the same instance. To use this scope, specify `SCOPE=instance`.
- **Method Scope** contains variables that should be unique to a method of a TeraScript class file. To use this scope, specify `SCOPE=method`.
- **Cookie Scope** contains variables that are sent to the user's Web browser as cookies (that is, a small text file kept by the Web browser for a specified amount of time). To use this scope specify `SCOPE=cookie`.
- **Custom Scope** is user-specified. It is outside of the scope search hierarchy.

Specifying Scopes

There are two methods of specifying a variable with a particular scope.

- Use the `SCOPE=`*scope* attribute.
- Leave out the `SCOPE=`*scope* attribute and specify a variable name as *scope*$*myvariable*; *scope* may be any valid scope specifier.

The behavior is undefined when both methods are used at once.

Scoping Precedence

When no scope is specified, TeraScript must find the variable by looking for the variable name within the various scopes. TeraScript has a set order in which it tries to find scopes. They are:

(in a TeraScript application file)
`request—»user—»application—»domain—»system`

(in a TeraScript class file)
`method–»instance–»request–»user–»application–»domain–»system`

Note Variable scoping precedence for variables and configuration variables does not check cookie scope.

For more information, see "domainScopeKey" on page 412.

If `domainScopeKey` resolves to empty for the current user, then domain is not checked. If there is no current application, application scope is not checked.

Variable Shortcut Description

There is a shortcut syntax for returning variables as well, with or without scope: use a double `@` and the name of the variable. The following two notations in each of the examples are equivalent:

```
<@VAR NAME="homer">
@@homer
```

```
<@VAR NAME="homer" SCOPE="domain">
@@domain$homer
```

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">
```

Configuration Variables

For a detailed list of configuration variables, see Chapter 3 of this manual.

TeraScript reserves special variables that contribute to the configuration of the server and also that provide default behaviors for users.

Configuration variables that control basic configuration of the server only exist in the system scope. Some configuration variables are valid in all scopes, or some scopes (for example, certain configuration variables are valid only in `application` and `system` scope); if so, they are subject to the full scoping mechanism described previously. Default values read from the preference file are stored in the system scope.

Examples

Accessing a request variable:

```
<@VAR NAME="foo" SCOPE="request">
<@VAR NAME="request$foo">
@@request$foo
```

Accessing a user variable:

```
<@VAR NAME="foo" SCOPE="user">
<@VAR NAME="user$foo">
@@user$foo
<@VAR NAME="foo" SCOPE="usr">
<@VAR NAME="usr$foo">
@@usr$foo
```

Accessing a system scope variable:

```
<@VAR NAME="foo" SCOPE="system">
<@VAR NAME="system$foo">
@@system$foo
<@VAR NAME="foo" SCOPE="sys">
<@VAR NAME="sys$foo">
@@sys$foo
```

Accessing a domain scope variable:

```
<@VAR NAME="foo" SCOPE="domain">
<@VAR NAME="domain$foo">
@@domain$foo
```

Accessing variable using scoping precedence:

```
<@VAR NAME="foo">
@@foo
```

Getting an array and formatting it for Results HTML:

```
<@VAR NAME="array">
```

Getting part of an array and formatting it for Results HTML:

```
<@VAR NAME="array[3,*]">
```

Getting an array and formatting it for Results HTML with attributes:

```
<@VAR NAME="array" APREFIX='<TABLE BORDER="2">'
ASUFFIX= </TABLE>' RPREFIX='<TR>' RSUFFIX='</TR>'
CPREFIX='<TD BORDER="2">' CSUFFIX='</TD>'>
```

Copying an array without formatting it (converting it to text):

```
<@ASSIGN NAME="array2" VALUE="<@VAR NAME='array'>">
```

Copying part of an array without formatting it:

```
<@ASSIGN NAME="array2" VALUE="<@VAR
NAME='array[*,4]'>">
```

Copying the formatted representation of an array to a variable:

```
<@ASSIGN NAME="array2" VALUE="<@VAR NAME='array'
FORMAT=text>">
```

Getting a document instance variable (XML) and performing no encoding on it:

```
<@VAR NAME="myDom">
```

Getting part of a document instance variable:

```
<@VAR NAME="myDom" XPOINTER="root().child(2)">
```

Copying a document instance:

```
<@ASSIGN NAME="myDom2" VALUE="<@VAR NAME='myDom'>">
```

Copying part of a document instance:

```
<@ASSIGN NAME="myDom2" VALUE="<@VAR NAME='myDom'
XPOINTER='root().child(2)'>">
```

See Also

Array-to-Text Attributes	page 17
<@ARRAY>	page 30
<@ASSIGN>	page 33
<@DEFINE>	page 100
Encoding Attribute	page 8
Format Attribute	page 11
variableTimeout	page 478
Working With Variables	page 319

<@VARINFO>

Syntax

<@VARINFO NAME=*varName* ATTRIBUTE=*attribute* [SCOPE=*scope*]>

Description

Returns information about variables and accepts four ATTRIBUTE values: TYPE, SIZE, ROWS, and COLS:

- TYPE returns either TEXT or ARRAY.
- SIZE returns the number of bytes used by the variable or array. For determining the number bytes, or characters, in a single text variable, this tag operates about twice as fast as <@LENGTH>.
- ROWS returns the number of rows if the variable is an array, or "0" otherwise.
- COLS returns the number of columns if the variable is an array, or "0" otherwise.

Examples

If the following variable assignments are made:

```
<@ASSIGN NAME="string" SCOPE="user" VALUE="abcdef">
<@ASSIGN NAME="array" SCOPE=user" VALUE="<@ARRAY
ROWS='5' COLS='3'>">
```

<@VARINFO> returns the following values:

```
<@VARINFO NAME="string" SCOPE="user"
ATTRIBUTE="type">
       (returns "TEXT")
<@VARINFO NAME="string" SCOPE="user"
ATTRIBUTE="size">
       (returns "6")
<@VARINFO NAME="string" SCOPE="user"
ATTRIBUTE="rows">
       (returns "0")
<@VARINFO NAME="string" SCOPE="user"
ATTRIBUTE="cols">
       (returns "0")
<@VARINFO NAME="array" SCOPE="user"
ATTRIBUTE="type">
       (returns "ARRAY")
<@VARINFO NAME="array" SCOPE="user"
ATTRIBUTE="rows">
       (returns "5")
```

```
<@VARINFO NAME="array" SCOPE="user"
ATTRIBUTE="cols">
```

(returns "3")

See Also

<@ASSIGN>	page 33
<@VAR>	page 292
<@VARNAMES>	page 300

<@VARNAMES>

Syntax

<@VARNAMES SCOPE=*scope* [{*array-to-text attributes*}]>

Description

Returns an array containing all variable names for a given scope.

For an explanation of the scoping rules, see <@VAR> <page $pagenum>.

The result array has one column and *n* rows where *n* is the number of variables in the specified scope.

There are array-returning attributes that can be used to specify prefixes and suffixes for the returned array, rows within the array, and columns within the rows. They are described in the section Array-to-Text Attributes <page $pagenum>. By default, the returned array is formatted as an HTML table.

Example

```
<@ASSIGN NAME="myVarNames" VALUE="<@VARNAMES
SCOPE='user'>">
```

Returns all the variable names for the current user scope as an array and assigns that array to the variable myVarNames.

See Also

<@ASSIGN> page 33
<@VAR> page 292

<@VARPARAM>

Syntax

```
<@VARPARAM NAME=varname [DATATYPE=datatype]
[SCOPE=scope]>
```

Description

The <@VARPARAM> Meta Tag is used to explicitly pass a value in the <@CALLMETHOD> Meta Tag. This Meta Tag instructs TeraScript Server to generate the appropriate binding call.

The NAME attribute is the name of a TeraScript variable to be used for parameter binding. The SCOPE attribute is an optional attribute defining the scope of the variable named in the NAME attribute.

The DATATYPE attribute is used only for COM Variants, and accepts the name of a COM data type to use (this is equivalent to using the parameter Type drop-down menu in the Call Method action). DATATYPE is ignored for non-Variant parameters.

Example

For examples of the use of <@VARPARAM> within the <@CALLMETHOD> Meta Tag, see <@CALLMETHOD> <page $pagenum>.

See Also

<@CALLMETHOD> page 55

<@VERSION>

Syntax <@VERSION [ENCODING=*encoding*]>

Description Returns the version number of TeraScript Server.

Example <@VERSION> returns the version number of TeraScript Server, for example, "6.0.3".

See Also

<@EDITION>	page 125
Encoding Attribute	page 8
<@PLATFORM>	page 221
<@PRODUCT>	page 224

<@WEBROOT>

Description

This Meta Tag returns the absolute path to the Web server document root.

<@WEBROOT> is useful for creating paths in File and External actions, which require absolute paths rather than paths relative to the Web server document root.

<@WEBROOT> does not include a trailing slash separator; this means that you must add one in certain cases.

For example, if your Web server document root on Windows corresponds to the root of a drive (D:), you must append a slash to create a well-formed path to that directory (<@WEBROOT>/); this is necessary if you want to read or write files to that directory.

Example

```
<@WEBROOT>
```

This Meta Tag returns the path to the Web server document root.

For example, if the Windows Apache Web server is installed to its default location, this tag returns: C:\Program Files\Apache Group\Apache2\htdocs\

```
<@WEBROOT><@APPFILEPATH>
```

These Meta Tags return the absolute path to where the current application file is located.

See Also

<@APPFILEPATH> page 24
<@APPPATH> page 27
<@CLASSFILEPATH> page 72

<@WHILE> </@WHILE>

Syntax

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

Description

The <@WHILE></@WHILE> Meta Tag block provides *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, <@CURROW> 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, a deprecated attribute [PUSH=*push*], allows the sending of data. This attribute has been deprecated as of the release of TeraScript 6. It is currently operational but will be removed in the next major version release. Developers are encouraged to discontinue use of this attribute. When applicable, a warning will be reported to the `witangoevents.log` file.

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

The </@WHILE> tag must always accompany <@WHILE> within the same action. Regular block nesting rules apply.

Example

```
<@ASSIGN name=request$Counter value="0">
<@WHILE EXPR="<@VAR request$Counter> < 5">
    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
```

See Also `<@FOR> </@FOR>` page 151

<@XSLT>

Syntax

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

Description

The <@XSLT> tag was removed in TeraScript version 6.1 and will return an error if used.

<@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.xsl’>”>
```

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.xsl'>" 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>s1 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>
```

<@!>

Syntax

<@! COMMENT=*comment*>

Description

Used to insert short comments in your application files. This tag is similar to the <@COMMENT> Meta Tag, except the comment goes inside the required attribute COMMENT, rather than between the <@COMMENT> and </@COMMENT> tags.

The tag and its contents are not returned the browser and any Meta Tags in the COMMENT attribute are not processed.

The attributes of this tag must obey all the quoting rules specified in Quoting Attributes <page $pagenum>; for example, if the COMMENT attribute contains spaces, you may not omit the quotes surrounding the attribute value.

Example

```
<@! COMMENT="Here is a comment.">
<@! "This code was written by Fred on 5/4.">
```

See Also

<@COMMENT> </@COMMENT> page 77
<@EXCLUDE> </@EXCLUDE> page 146

<@!>

CHAPTER 3

Custom Meta Tags

3

A Guide to Custom Meta Tags

TeraScript allows the use of *custom Meta Tags*, which are user-defined TeraScript Meta Tags that map directly to any method call supported by TeraScript, and can be used to call TeraScript class files, COM objects, and JavaBeans from TeraScript application files and TeraScript class files.

Custom Meta Tags can apply to all of TeraScript Server (system scope) or be specific to an application (application scope). Custom Meta Tags can be shared with other developers by distributing the tag definition file and any objects called by the custom Meta Tag.

This chapter describes using custom Meta Tags, creating a file that defines one or several custom Meta Tags, and the process of installing custom Meta Tags.

Using Custom Meta Tags

Once a custom Meta Tag has been created and installed on your system, all a user needs to do is type in the custom Meta Tag with any required attributes in any place where Meta Tags can be specified. The object instance is created, and a method call to the specified object is made when TeraScript Server executes the file. There are some issues to be aware of when using custom Meta Tags, described in this section.

Attributes of Custom Meta Tags

Required attributes of a custom Meta Tag must be specified when that custom Meta Tag is used; otherwise, TeraScript generates an error (naming the first missing attribute), and execution ends.

For more information, see "Encoding Attribute" on page 8 and Format Attribute on page 11.

All custom tags have two standard attributes: `FORMAT`, and `ENCODING`. Returned values are encoded according to the `ENCODING` attribute, and formatting according to the `FORMAT` attribute, if either or both of these attributes is specified when the custom Meta Tag is used in an application file or class file.

Tag Name Conflicts

Application-specific custom Meta Tags can share names with system scope custom Meta Tags, in which case the application scope tag is used. Within a particular scope—application or system—tag names must be unique; TeraScript generates a warning in the event log and uses the first tag when a duplicate name is encountered within a scope.

Caution If a custom Meta Tag has the same name as a built-in TeraScript Meta Tag, the custom Meta Tag does not work; the TeraScript Meta Tag takes precedence. To reduce the chance of this happening consider using an underscore "_" in your custom tag name.

Custom Meta Tag Limitations

Custom Meta Tags do not support blocks. This means that custom Meta Tags can have attributes but no content because they can not have end tags.

Tags can only call objects that are defined within the current tag definition file.

Creating Custom Meta Tags: Tag Definition File

You define custom Meta Tags in *tag definition files*, which are XML files. These files must reside in a specific directory for system scope custom Meta Tags, and specific directories for application-specific custom Meta Tags (different directories for each application).

For more information, see customTagsPath on page 399.

The `customTagsPath` configuration variable, available in system and application scope, defines the path to the directories where tag definition files reside.

A tag definition specifies a custom tag and the object and method to call for the custom tag. A custom tag definition file contains one or more tag packages, which each contain at least one object specification and at least one tag definition. A tag package groups related tags and shares objects with other tag packages.

You define custom Meta Tags by creating tag definition files based on a specific XML document type definition (DTD) developed by Tronics Software.

Custom Tag Definition File Format

The format of the XML custom Meta Tag definition file is given below.

Note XML is case-sensitive. The names of the XML elements in tag definition files must be capitalized exactly as shown (for example, `<packages>`).

The following is an annotated custom Meta Tag definition XML file, describing each element:

```
<tagpackages>                                          root

<packagedef ID=required; defines package with unique identifier >

    <author>optional; author of package</author>

    <version>optional; version of package</version>

    <copyright>optional; package copyright info</copyright>

    <packagedesc>optional; description of package</packagedesc>


    <objects>                                          specifies settings

          <objectdef
```

ID=required; a name you choose to uniquely identify the object used in the tag definition.

type=required; type of object (COM, JavaBean, or TCF [TeraScript class file])

systemobject=optional; default is false; if true, TeraScript uses an existing instance instead of creating a new one (COM objects only).

>

<name>

required; ProgID or ClassID (for COM); TeraScript class file name; or JavaBean name. This is the same name that must be specified in the <@CREATEOBJECT> Meta Tag. For COM and JavaBean methods, the method name is case-sensitive.

</name>

<varname>

optional; when specified, TeraScript uses this object instance variable. If you want the tag processing to create the object instance named here, you must specify the scope and name elements and the TYPE and ID attribute of <objectdef>. If you want to be responsible for creating the instance before calling the custom tag, you need only specify the <scope> element. If the named object instance variable does not exist, it is created.

If <varname> is not specified, a temporary variable is used to create the instance of the object called by the custom Meta Tag; this instance goes away immediately after processing of this tag has finished.

<varname> **may contain TeraScript Meta Tags.**

</varname>

<initstring>

optional; used to create monikers for COM objects

</initstring>

<scope>

scope in which object instance is created or obtained from (application, domain, user & request are allowed). May contain TeraScript Meta Tags.

</scope>

</objectdef>

</object>

<tagdef

name= tag name; must begin with a letter and may contain letters, numbers or underscores; no leading '@' is required.

objectid=identifies object (ID of an OBJECT element in the same package) whose method is called by this tag.

methodtype=(JavaBeans and COM only): INVOKE (default), SET or GET.
>

`<method>` **name of method to call** `</method>`

`<encoding>`

optional; determines the encoding used for the return value of the tag; if NONE is specified, value is not encoded, even if the tag is used in Results HTML. If element is not specified, TeraScript encodes the return value as it does other tags, determined by context; special characters are encoded if in Results HTML. Can be overridden by the ENCODING attribute of the custom Meta Tag.

`</encoding>`

`<tagdesc>`

optional; describes tag for editing environments

`</tagdesc>`

zero or more attribut

`<attrdef`

name=name of custom tag attribute

required=optional; TRUE or FALSE (default) determines whether this attribute may be omitted when tag is used >

`<defaultvalue>`

sets an optional default value for the attribute; ignored if required set to TRUE

`</defaultvalue>`

`<attrdefdesc>`

optional description of the attribute

`</attrdefdesc>`

`</attrdef>`

`</tagdef>`

`</tags>`

`</packagedef>`

`</tagpackages>`

All leading and trailing whitespace between and within start and end tags in the custom tag definition files is ignored.

Note TeraScript Meta Tags are supported only in the above-noted elements of tag definition files; elsewhere, they cannot be used.

Loading Tags

TeraScript Server loads tag definition files from the directories defined by the `customTagsPath` configuration variable. All files in the directory specified by customTagsPath, and any subdirectories, are loaded.

For more information, see startupUrl on page 461.

Tag definition files for system scope load before the URL defined by the `startupURL` configuration variable; this means that custom Meta Tags are available for use in the application file called by the startupURL configuration variable.

Reloading Custom Meta Tags

For more information, see "<@RELOAD-CUSTOMTAGS>" on page 231.

The <@RELOADCUSTOMTAGS> Meta Tag forces a reload of the specified scope's custom tags file. The default value of the `scope` attribute is `system`. This tag requires that the user scope `configPasswd` match the `configPasswd` configuration variable for the scope requested (system or application).

Retrieving Information on Custom Meta Tags

For more information, see "<@CUSTOMTAGS>" on page 88.

The <@CUSTOMTAGS> Meta Tag returns an array of all custom Meta Tags in the specified scope. The names of the array's columns (name, package, and scope) are put into row 0 of the array. No password is required to use this tag.

Installing Custom Meta Tag Definition Files

Follow these steps to install custom Meta Tag definition files:

- Install the object(s) that your custom Meta Tags call

  Follow your operating system instructions for installing objects or refer to the object vendor's documentation for installation.

- Copy the custom tag definition XML file(s) to the folder pointed to by `customTagsPath` configuration variable; by default, this is the `CustomTags` folder under the configuration folder. You may need to set the `customTagsPath` configuration variable to point to a different folder. If TeraScript Server is already running, load your custom Meta Tags by running an application file containing the `<@RELOADCUSTOMTAGS>` Meta Tag, or restart TeraScript Server.

Application-specific Custom Meta Tags

In order to use application-specific custom Meta Tags, the system administrator must set the value of the `customTagsPath` configuration variable in application scope when setting up an application. Application-specific custom Meta Tags are loaded when that application is started; that is, when a TeraScript application file in the application is called.

Custom Meta Tag Example: tabletag.xml

You create custom Meta Tag definition XML files based on the annotated custom Meta Tag definition file (see Custom Tag Definition File Format on page 313). This section describes an example of the files necessary to create and use a custom Meta Tag.

1. Defining the Custom Meta Tag

`tabletag.xml` contains the following XML:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE TAGPACKAGES SYSTEM "ctags.dtd" >
<tagpackages Version="0x02000001">
<packagedef ID="MyPackage">
   <objects>
      <objectdef ID="arrayutils" type="TCF"
                  systemobject="false">
         <name>array_utils.tcf</name>
         <varname>arrayutilsinstance</varname>
         <scope>request</scope>
      </objectdef>
   </objects>
   <tags>
      <tagdef name="table" objectid="arrayutils">
         <method>format_array</method>
         <encoding>NONE</encoding>
         <attrdef name="the_array"required="true">
         </attrdef>
         <attrdef name="border" required="false">
         </attrdef>
         <attrdef name="cellspacing" required="false">
         </attrdef>
         <attrdef name="cellpadding" required="false">
         </attrdef>
         <attrdef name="height" required="false">
```

```
            </attrdef>
            <attrdef name="width" required="false">
            </attrdef>
            <attrdef name="bgcolor" required="false">
            </attrdef>
            <attrdef name="bgcolor2" equired="false">
            </attrdef>
        </tagdef>
    </tags>
    </packagedef>
</tagpackages>
```

`tabletag.xml` defines the <@TABLE> custom Meta Tag with the following syntax:

```
<@TABLE the_array=arrayname [BORDER=num]
[CELLSPACING=num] [CELLPADDING=num] [HEIGHT=num/
percent] [WIDTH=num/percent] [BGCOLOR=color]
[BGCOLOR2=color]>
```

2. Installing the Custom Meta Tag

The custom tag definition file must be put into the folder specified by the `customTagPath` configuration variable, for either application or system scope.

To use the <@TABLE> custom Meta Tag in your application files, you load the custom Meta Tag definition file by restarting TeraScript Server or executing the <@RELOADCONFIG> tag; in the case of applications, the tag definition file for that application is loaded when that application is created.

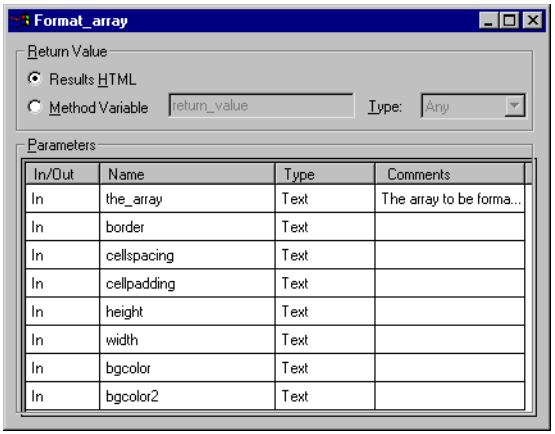
3. Installing the Object

The <@TABLE> custom Meta Tag calls a TeraScript class file named `array_utils.tcf`.

For more information, see "TCFSearchPath" on page 464.

The `array_utils.tcf` class file must be installed in a directory where TeraScript Server looks for TeraScript class files. These directories are specified using the `TCFSearchPath` configuration variable.

The parameters of `Format_array` from `array_utils.tcf` are shown below.



These parameters map to the attributes displayed in the `table.taf` Results HTML:

- `the_array` is the name of a variable (for example, `request$myArray`) containing an array to be returned as an HTML table.
- The next five attributes are straightforward; they are included in the resulting `<@TABLE>` custom Meta Tag.
- The `BGCOLOR` attribute, if included, is used for the tables background color. If `BGCOLOR2` is also specified, you get a table whose rows alternate between `BGCOLOR` and `BGCOLOR2`.

The `Format_array` Results HTML is processed by TeraScript Server, and the results are displayed in the `table.taf` Results HTML, which is read by the Web browser when `table.taf` is executed.

The `Format_array` Results HTML from `array_utils.tcf` looks like this:

```
<@EXCLUDE>
    <@IF expr="len(@@method$bgcolor) and
    len(@@method$bgcolor2)">
        <@ASSIGN method$alternate 1>
    <@ELSE>
        <@ASSIGN method$alternate 0>
    </@IF>
</@EXCLUDE>
<TABLE
```

```
    <@IFEMPTY "@@method$border">
    <@ELSE> border=@@method$border</@IF>
    <@IFEMPTY "@@method$cellspacing">
       <@ELSE> cellspacing=@@method$cellspacing</@IF>
    <@IFEMPTY "@@method$cellpadding">
       <@ELSE> cellpadding=@@method$cellpadding</@IF>
    <@IFEMPTY "@@method$height">
       <@ELSE> height=@@method$height</@IF>
    <@IFEMPTY "@@method$width">
       <@ELSE> width=@@method$width</@IF>
    <@IF expr="!(@@method$alternate) and
       len(@@method$bgcolor)" >
       bgcolor=@@method$bgcolor</@IF>
>
<@ROWS array=@@method$the_array>
<TR ALIGN=center
    <@IF "@@method$alternate and ((<@currow> % 2) !=
    0)">
          BGCOLOR=@@method$bgcolor
    <@ELSEIF expr="@@method$alternate">
          BGCOLOR=@@method$bgcolor2
    </@IF>
>
<@COLS>
    <TD><@COL></TD>
</@COLS>
</TR>
</@ROWS>
</TABLE>
```

4. Using the Custom Meta Tag in a TeraScript Application File

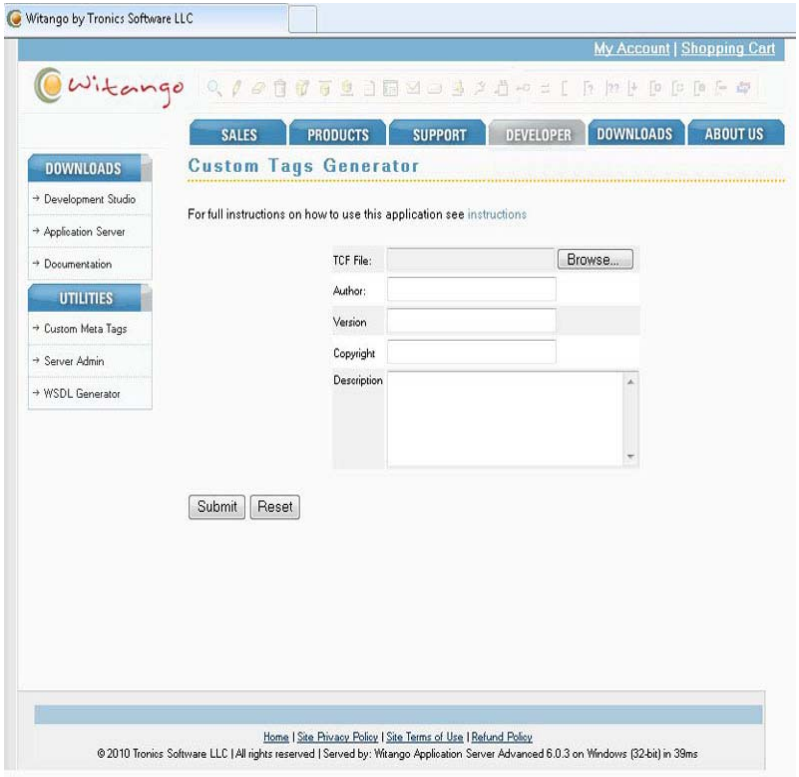
The TeraScript application file, `table.taf`, contains an Assign action and Results HTML containing the `<@TABLE>` custom Meta Tag

You could modify this example to allow users to specify other table attributes to be passed through to the TeraScript class file in the <@TABLE> custom Meta Tag.

Custom Tag Generator

A tool to produce Custom Definition Files is available on-line on from the TeraScript web site, http://www.terascript.com. The tool is located on the Developer page (click on the Developer tab) under the “Utilities” heading. Or, go to http://www.terascript.com/developer/custommetatags. Here you will find instructions on how to use the custom tag generator, and, access to the tool itself.

The output of the Custom Tag Generator is a tag definition file in XML format.



CHAPTER 4

Working With Variables

4

Using Variables in TeraScript

Variables are containers that you can assign a value to; they are created and assigned values using the Assign action or the <@ASSIGN> Meta Tag. See "<@ASSIGN>" on page 32. Optionally, you can create an empty variable for use later using the <@DEFINE> Meta Tag. For more information, see "<@DEFINE>" on page 98

Every variable belongs to a *scope*, which tells TeraScript if the variable is accessible to the particular application file execution, within a TeraScript application, for a user, or for a particular domain being served with TeraScript. Variables can also belong to special scopes within TeraScript class files that apply to a method or an instance of a TeraScript class file.

Arrays are a special variable type that allow you to create a structured data table with multiple values, as opposed to standard variables which only store one value.

You can also create variables that contain XML data structures (document instance variables) and variables that contain objects.

One important set of variables determines the behavior of certain TeraScript options. These are called configuration variables.

This chapter covers the following topics:

- introduction to variables—standard and array—including variable scope and its effects
- how to assign values to variables
- configuration variables
- the user key.

About Variables

Variables are defined and given values with an Assign action in a TeraScript application file or a TeraScript class file.

Variables can also be assigned values by using the <@ASSIGN> Meta Tag.

You can assign any combination of literal values and Meta Tag values to a variable.

For example, to assign to a variable a combination of Meta Tags that could evaluate to a full phone number using values from area code and phone number form fields, you could use the following Meta Tags:

```
<@ASSIGN NAME=PhoneNum
VALUE="(<@POSTARG area>) <@POSTARG phone>">
```

This assigns to a variable called PhoneNum (in default scope). The variable is created, if it does not already exist, and the value of the variable is set to what the Meta Tags within the VALUE attribute evaluate to when the TeraScript application file is executed, plus the characters within the double-quotes before and after the Meta Tags.

Naming Variables

All variable names:

- must start with a letter
- may contain numbers, letters, and the underscore ("_") character
- may be no longer than 31 characters.

Variable names are case insensitive; for example, myVar is the same variable as MYVAR and MyVaR.

Variable Types

Variables can be Text, Arrays, DOMs or Objects. Details of these types are set out below.

Text

Which is used to reference a text string.

Arrays

Which is used to reference an array. For more information see Arrays on page 336.

DOM (XML document instance)

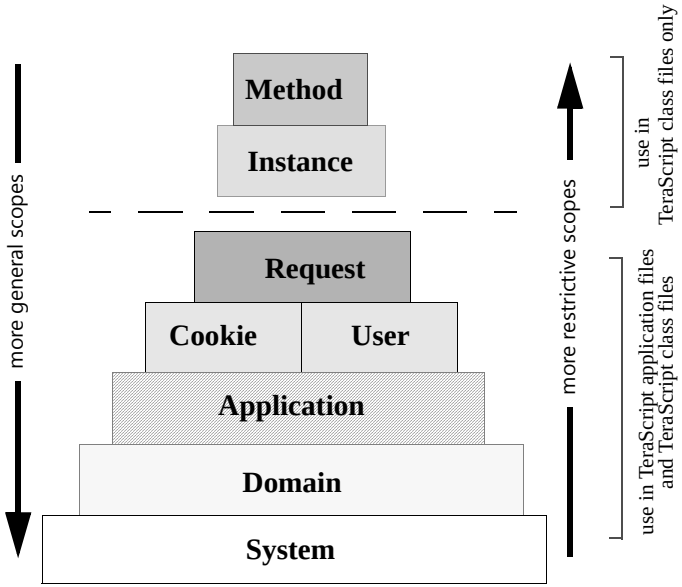
`Which is` used to reference a document instance (XML) variable. For more information see Document Object Model on page 343.

OBJECT

Which is used to reference a variable of an object instance.

Understanding Scope

Every variable belongs to a *scope*. There are several scopes that are used for different purposes within TeraScript. The following diagram shows the different scopes and their relationships.



Variables that are assigned values in more restrictive scopes override variables with the same name that are assigned values in more general scopes.

Six scopes are available to all TeraScript files (TeraScript application files and TeraScript class files).

Within these six scopes:

- **Request scope** is the most restrictive: variables that belong to this scope are used only for the execution of a particular TeraScript application file.
- **Cookie scope** refers to variables that are sent to and kept by Web browsers.
- **User scope** refers to variables that are set for particular users within TeraScript.

- **Application scope** refers to variables that apply to all TeraScript application files in a particular TeraScript application.
- **Domain scope** refers to variables that are used in a particular TeraScript domain, which is a single domain name (base URL or IP address) or a defined group of domain names (a TeraScript domain).
- **System scope** is the most general, applying to the entire TeraScript Server. This scope is restricted to configuration variables.

In addition to these basic TeraScript scopes, you can also define custom scopes.

When you use TeraScript class files, two additional scopes are available to you. The variables in these scopes are only available in methods within TeraScript class files.

- Method scope refers to variables that are used only in the current invocation of the method.
- *Instance* scope refers to variables that are used in the instance of the TeraScript class file to which the current method belongs.

You can find more details on these scopes in the following sections.

Basic TeraScript Scopes

Request Scope

Request scope is used for variables that expire after an application file is executed. That is, after the application file (and its branches, if any) has finished executing, the variable is purged from memory.

Request scope is created at the beginning of the TeraScript application file execution, persists in any TeraScript application files branched to and TeraScript class file methods called, and is destroyed when the TeraScript application file ends execution and returns results to the user.

An example of the use of a request variable is a variable for a counter within a loop. The counter is not needed outside the application file that it is in, so using a request variable is appropriate.

For another example, a user might enter a choice during the execution of an application file to have full explanations returned during the execution of an application file (for a beginning user), or rather terse commands (for an experienced user). At the beginning of the application file, a request variable is set, and that variable is used in the rest of the application file to determine what HTML is returned to the user.

Cookie Scope

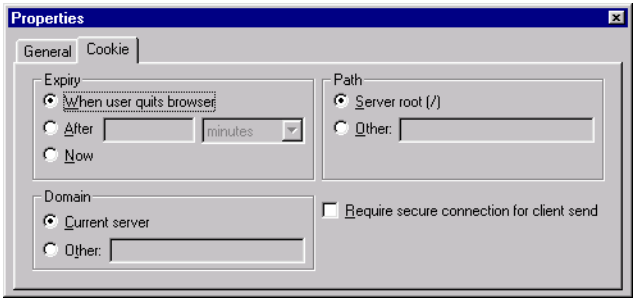
Cookie scope is used for variables that are sent to the user's Web browser as cookies (that is, saved in a small text file and kept by the Web browser for a specified amount of time).

Cookies are saved by the Web browser and then are sent to the site by the Web browser when it returns to the site that generated the cookie in the first place. Cookie variables can be used to save state information between Web browser sessions or in a single Web browser session.

There are a variety of cookie options that can be set for cookie variables, such as when the cookies expire.

Note TeraScript only supports cookies up to 3071 bytes in length. Cookies that are set which are longer than 3071 bytes will be truncated.

The default values for new cookies are as shown:



- **Expiry**

  When user quits browser is the default cookie behavior as described in the cookie specifications. When this option is chosen, the `Expires` value is omitted from `Set-Cookie` line in the cookie header.

  After __ [time units]. The drop-down menu for time units has minutes, hours, days, and years options. The text entry field holds up to 31 characters; a Meta Tag can be specified there.

- **Domain**

  Current server omits the Domain value from the `Set-Cookie` line, causing the cookie to be valid for the current server.

Other allows specification of any domain string up to 31 characters. ".example.com", for example, would cause the cookie to be sent back to www.example.com, demo.example.com, sales.example.com, and so on.

- **Path**

  Server root (/) specifies that the cookie be sent for all paths within the specified domain.

  Other allows specification of a path string up to 31 characters. For example, /TeraScript/ would cause the cookie to be sent back only for URLs below the TeraScript folder.

- **Require secure connection for client send**

  True (enabled) or **False** (disabled). This option sets the Secure value of the Set-Cookie line. If the value is set to true, then the cookie is sent back by the Web browser only if a secure connection is being made.

User Scope

Variables that have *user* scope let you store and access values associated with a particular user. Once an assignment has been made to a user variable, its value is available in subsequent actions within the same application file execution and in any application files executed afterward.

For example, user variables would be necessary in an application file that logs users in to a private area of your Web site, displaying a Web page that prompts for a user's name and password. You might want to save the entered name so you can display it on personalized pages in subsequent queries. Or you may need to use the name to restrict database queries performed by that user. Both of these operations would be performed with the use of a user variable.

For more information, see "variableTimeoutTrigger" on page 479.

User variables expire 30 minutes after the user associated with them last accesses TeraScript. You can change the expiry time by changing the value of the TeraScript configuration variable variableTimeout in user scope.

Application Scope

Application scope defines variables that apply to all application files within a TeraScript application. When TeraScript checks for variables, application scope is less restrictive than user scope but more restrictive than domain scope.

Applications are defined as a group of TeraScript application files in a particular *application folder*, which is defined in the application configuration file or configured with the TeraScript Administration Application (the `config.taf` application file).

You can define many configuration variables in application scope; they apply only to the TeraScript applications.

Because application scope is more restrictive than domain scope, if you want a certain application scope to apply to all domain names in a particular domain, you must specify the TeraScript domain when you specify the TeraScript application.

Application Scope variables do NOT timeout.

You can turn application scope on or off using the `applicationSwitch` configuration variable. You improve performance by setting this switch to `off` when applications are not being used.

Domain Scope

Domain scope is used for variables that are relevant to a particular domain name or TeraScript domain. Domain variables, like system variables, are persistent across applications, application files, and users.

To explain domains, it is useful to distinguish between *domain names* and *TeraScript domains*.

Domain names are different base URLs and IP addresses: for example, one Web server could be hosting several different domains (`www.my_company.com`, `www.your_company.com`, `www.a_non_profit.org`) at several different IP addresses (for example, `152.23.23.45`, `152.65.34.32`, and so on).

For more information, see domainScopeKey on page 412.

By default, the *domain key* (that is, the piece of information TeraScript uses to determine which domain a user belongs to) is based on the `SERVER_NAME` CGI parameter. The value of the `SERVER_NAME` parameter comes from the URL used to initiate each request by a user. TeraScript by default uses an encrypted form of the domain name used to access a TeraScript application file—base URL or IP address—as the domain scope key, and any variable set with `scope=DOMAIN` is set for the particular domain name where the application file is executed. For example, if a user requests

```
http://www.yourserver.com/foo.taf
```

the domain variable is keyed on "`www.yourserver.com`". Everyone using "`www.yourserver.com`" to access a TeraScript application file gets the same values for domain variables.

However, if a user accesses the exact same application files with the IP address of the Web server (`http://206.186.95.106/foo.taf`), the domain scope variable is different because the key value is now the IP address, and not the domain name. The same applies if you access the Web site locally with `http://localhost/foo.taf`.

There are cases where you want the same domain variables to be available even though a user is hitting your Web site through different domain names (base URLs or IP addresses).

You can set up a *TeraScript domain* which incorporates several domain names and IP addresses.

TeraScript domains are listed in the domain configuration file. To set up a TeraScript domain, use the Administration Application `config.taf` application file to specify which domain names should be part of the TeraScript domain. Depending on how users access your TeraScript Server, you may need to set up one or several TeraScript domains. You may also need to list `localhost` and `127.0.0.1` as domain names that belong to a particular TeraScript domain, so that hits coming in from those domain names (which always reference the current machine) are part of a TeraScript domain, and share domain scope variables correctly.

Domain scope variables do not timeout.

System Scope

System scope is used for variables that are set at the system level, that is, only configuration variables. Many configuration variables that affect the behavior of TeraScript are set with system scope. For example, the variable `dateFormat` sets the way the date is displayed when it is returned by TeraScript with a Meta Tag such as `<@CURRENTDATE>`. If this variable is set with system scope, then all dates returned by TeraScript are in that format.

For more details on different scopes for configuration variables, See "Understanding Scope" on page 327.

Certain configuration variables can be set for different scopes. This means that the value changes in one particular scope (and all scopes that scope dominates) while maintaining its default value elsewhere.

For example, `dateFormat` can be set with `scope=LOCAL`, which changes the format of the date for the current application file; `scope=USER`, which would change the format of the date for all application files executed by a particular user; or `scope=DOMAIN`, which would change the format of the date for all application files and users that access TeraScript from a particular URL or TeraScript domain.

To set system configuration variables

.

You can set system configuration variables with the TeraScript Administration Application (the `config.taf` application file). This application file prompts you for a password and then presents groups of related configuration variables on different screens, allowing you to easily set system configuration variables.

To set configuration variables without using the Administration Application (the `config.taf` application file)—that is, in an application file—you must know the correct password. This password is stored in the `t4server.ini` configuration file and a configuration variable called `configPasswd`.

When you attempt to set a system configuration variable, TeraScript checks to see if there is a variable called `configPasswd` with `scope=USER` that matches `configPasswd` with `scope=SYSTEM`. If there is, TeraScript lets you change configuration variables. If not, TeraScript returns an error message.

Setting `configPasswd scope=USER` can interact with user keying mechanisms;

That is, you must assign the value of an entered string (for example, `<@POSTARG NAME="Password">`) to the configuration variable `configPasswd` with `scope=USER` using the Assign action or the `<@ASSIGN>` Meta Tag.

Note You do not need this password to get the values of system configuration variables, except for `configPasswd` itself.

TeraScript Class File-only Scopes

Instance Scope

Instance scope is available only when using object instances of TeraScript class files. A variable in this scope persists across all calls to methods in the same object instance.

TeraScript Server creates this scope when it creates an object instance. This scope goes away when TeraScript Server destroys the object instance (more precisely, after the TeraScript class file calls the `On_Destroy` method).

Method Scope

Method scope is available only in TeraScript class file methods. A variable in this scope exists for the duration of a single Call Method action. All parameters are part of method scope.

TeraScript Server creates this scope when it begins to execute a TeraScript class file method. This scope goes away when the method returns.

Custom Scopes

In addition to specifying variables that apply to methods, instances, TeraScript application files, users, applications, domains or to all of TeraScript Server, TeraScript allows you to create custom scopes that are used to store variables.

Custom scopes are useful for values that should be available everywhere, to all users.

Custom scopes are less restrictive than other scopes because they apply to all of TeraScript Server, regardless of the domain name.

Custom scopes fall outside of the TeraScript scope hierarchy; therefore, only explicitly specifying the scope when returning the variable allows TeraScript Server to find the variable. You must specify a custom scope to access a variable assigned to it; custom scopes are not searched when variables are assigned or referenced without scope.

Example: Setting Up a Chat Room

You are creating an application file to set up a chat room for all users of your TeraScript Server — no matter what application they are currently in. You create a series of variables in custom `chat` scope to set up your chat server; for example, a variable called `chat$allusers` that stores a list of current "chatters". All variables in `chat` scope can be accessed by all users of your TeraScript Server who execute application files that specify the custom `chat` scope.

Specifying Custom Scope

You can specify a custom scope anywhere TeraScript accepts a scope; for example, you can create a custom scope in the text field/drop-down menu in the Assign action. The following are examples of how custom scope is used:

```
<@ASSIGN NAME="foo" SCOPE="myCustomScope">
<@VAR NAME="myCustomScope$foo">
```

Timeout for Custom Scope

For more information, see "variableTimeoutTrigger" on page 479.

`variableTimeout` can be used to set the timeout value of a custom scope. `variableTimeout` allows you to specify the expiration interval (in minutes) of custom scope variables.

By default, `variableTimeout` of a custom scope is set to the timeout of user scope; that is, 30 minutes.

To set `variableTimeout` for a custom scope, assign a value to `variableTimeout` in that custom scope; for example:

```
<@ASSIGN NAME=variableTimeout
SCOPE="myCustomScope">
```

`variableTimeoutTrigger` allows you to specify an HTTP URL to activate just prior to expiry of the custom scope's variables. This configuration variable can be used for a variety of purposes, for example, to clear the database of expired variables used in custom scope.

For more information and the format and restrictions of this value, see variableTimeoutTrigger on page 479.

There is no default timeout trigger for `variableTimeoutTrigger`.

Configuration Variables and Custom Scope

You can enable and disable custom scope using the `customScopeSwitch` configuration variable. Applications that use custom scope do not work when this switch is disabled. The default setting for `customScopeSwitch` on TeraScript Server is `off`.

For more information, see customScopeSwitch on page 398.

Returning Variable Values

To have TeraScript return the value of a particular variable, use the <@VAR> Meta Tag. This tag is replaced with the variable value at the time that the application file is executed. For example, to return the value of the variable named `fred` in user scope, use the following Meta Tag and scope attribute:

```
<@VAR NAME="fred" SCOPE="user">
```

Default Scoping Rules

If you do not specify a scope attribute, TeraScript checks for matching variables in different scopes, from the most restrictive to the most general. For example, if you use the following Meta Tag:

```
<@VAR NAME="fred">
```

TeraScript checks for matching variables in request, user, application, domain, and system scope.

Note Cookie scope is a special scope, and TeraScript does not check for matching variables with this scope when returning unscoped variable values with <@VAR>.

TeraScript returns the value for the first matching variable that it finds in the scope hierarchy.

Caution TeraScript does not search any custom scope. If you want to assign a value to or retrieve a value from a custom scope, you must explicitly specify the scope.

Shortcut Syntax for Returning Variables -@@request$foo

There is a shortcut syntax for returning variables: use a double "@" and the scope and name of the variable. Use a scope parameter with this shortcut, place it in front of the variable being accessed, with a dollar sign ("$") in between. The following two notations are equivalent:

```
<@VAR NAME="fred" SCOPE="user">

is same as

   @@user$fred
```

If you are creating complicated Meta Tag syntax, using this shortcut may help to make things clearer.

The 31-character length limit on variable names is exclusive of any scope specifier; for example, in `local$longvariablename`, the length limit applies to the variable name portion.

Purging Variables

You can clear obsolete or no-longer-needed variables from memory by using the <@PURGE> Meta Tag to remove a particular variable from a scope or to remove all variables from a scope.

For more information, see "<@PURGE>" on page 223.

For example, using <@PURGE> to remove the variable `foo` from `DOMAIN` scope looks like this:

```
<@PURGE NAME="foo" SCOPE="domain">
```

Clearing a user's variables when they log out is one example of using the <@PURGE> Meta Tag to remove all variables from a specific scope. The following example shows how to remove all variables from `USER` scope:

```
<@PURGE SCOPE="user">
```

Arrays

Arrays are a special type of variable that allow you to store many different values in a structured format. This is distinct from the standard variable, which only stores one value.

Arrays are structured as a table with rows and columns; values are saved at each row and column intersection. This is similar to the way values are saved in a database table: as rows and columns.

For example, a three-row by four-column array with the values of the first twelve integers looks like this:

```
1  2  3  4
5  6  7  8
9  10 11 12
```

Setting Arrays

For more information see <@ARRAY>page 29

To create an array—for example, the one in the previous paragraph—use the <@ARRAY> Meta Tag within the Assign action or the <@ASSIGN> and <@ARRAY> Meta Tags together:

You can use the <@ARRAY> tag to create arrays by using only rows and columns (ROW attribute and COLS attribute), which creates an array with the specified dimensions, all of whose elements are empty, or by using the VALUE attribute to create and initialize the array with values. If ROWS, COLS, and VALUE are specified, they must match in terms of the number of rows and columns specified.

The Meta Tag assignment of an array to a variable looks as follows:

For more information see <@ASSIGN>page 32

```
<@ASSIGN NAME="arrayVar" VALUE=<@ARRAY ROWS="3"
COLS="4" VALUE="1,2,3,4;5,6,7,8;9,10,11,12">>
```

When using an Assign action to create an array and assign it to a variable, the **Value** field would contain an <@ARRAY> Meta Tag.

For more information, see cDelim on page 389 and rDelim on page 453.

(The row and column delimiters for VALUE are set with the configuration variables rDelim and cDelim, or with the <@ARRAY> tag's optional attributes RDELIM and CDELIM. By default, they are set to ";" and "," respectively.)

Array cells can contain single values only. They cannot contain other arrays.

Array Formats

How arrays are returned depends on context, that is, where an array-returning Meta Tag such as <@VAR> is used.

Arrays are returned as array variables (with their special internal structure) when used in assignments to other variables, for example:

```
<@ASSIGN NAME="fred" VALUE="<@VAR NAME=
'array_variable'>">
```

When referred to anywhere else (for example, returned in Results HTML using the <@VAR> Meta Tag), arrays are converted to a text representation using the supplied array-returning attributes for prefixes and suffixes, or the configuration variable defaults, if no attributes are specified.

Returning the Values of Arrays

An array is *not* just a list of values; it has two-dimensional structure. For example, if you set up and give values to the variable `fred` as above and then ask TeraScript to return the value of the array in HTML only, TeraScript returns the structured string of values, using delimiters to separate rows and columns.

For more information, see Array-to-Text Attributes on page 16.

Along with many other uses for programming and database work, arrays offer a quick method of returning a table or ordered list of values in HTML. All Meta Tags that return arrays, such as <@VAR>, have attributes that can be used with arrays:

`APrefix`: the array prefix string
`ASuffix`: the array suffix string
`RPrefix`: the row prefix string
`RSuffix`: the row suffix string
`CPrefix`: the column prefix string
`CSuffix`: the column suffix string

The defaults of these parameters are set with configuration variables. If the above array variable is returned using <@VAR> with the default parameters:

```
APrefix='<TABLE BORDER=1>'
ASuffix='</TABLE>'
RPrefix='<TR>'
RSuffix='</TR>'
CPrefix='<TD>'
CSuffix='</TD>'>
```

For more information on arrays, see <@ARRAY> on page 29.

The following simple table is returned when <@VAR NAME="fred"> is retrieved in Results HTML:

```
<TABLE BORDER=1><TR><TD>1</TD><TD>2</TD><TD>3</TD>
<TD>4</TD></TR><TR><TD>5</TD><TD>6</TD><TD>7</TD>
<TD>8</TD></TR><TR><TD>9</TD><TD>10</TD>
<TD>11</TD><TD>12</TD></TR></TABLE>
```

The rendered HTML code displayed in a Web browser looks like this:

1	2	3	4
5	6	7	8

9	10	11	12

Using different suffixes and prefixes could create different kinds of HTML code, such as various kinds of lists.

You can retrieve one single value from an array by specifying row and column co-ordinates:

```
<@VAR NAME="FRED[2,3]">
```
⇒ TeraScript returns `7`

You can return one column or one row from an array by specifying only one of the co-ordinates, and setting the other to *:

```
<@VAR NAME="FRED[2,*]">
```
⇒ TeraScript returns a one-column array with the values `5 6 7 8`

```
<@VAR NAME="FRED[*,3]">
```
⇒ TeraScript returns a one-row array with the values `3 7 11`

Special Array: resultSet

Whenever an action returns a result rowset in TeraScript (for example, a Search action), that rowset is assigned, in array format, to the request variable `resultSet`. This variable could be used in Results HTML to return your search result, for example:

```
Your search returned the following results:
<@VAR NAME="resultSet" SCOPE="request">.
```

As with any array variable, you can use the prefix and suffix attribute name/value pairs to change how the array is formatted in HTML.

resultSet Named Columns

A special property of `resultSet` is that when it returns the results of a Search action or Direct DBMS query, its columns are *named*. This means that you can refer to the names of columns (instead of the column number) when returning the value of the `resultSet` array. For example, if you selected these columns in the order shown (as the result of a Search action):

customer_last_name	customer_ID	phone_number
Smith	8099	555-1155
Jones	3334	555-1454
Johnson	1234	555-0023

The following expressions in Results HTML evaluate as shown:

```
<@VAR NAME="resultSet[3,2]">
```
⇒ TeraScript returns `1234`

Using named columns, you could return the same values using the following syntax:

```
<@VAR NAME="resultSet[3,customer_ID]">
```
⇒ TeraScript returns `1234`

You can use a combination of asterisks and named columns:

```
<@VAR NAME="resultSet[*,customer_last_name]">
```
⇒ TeraScript returns `Smith Jones Johnson` (a one-column array).

Row Zero of Arrays

Many returned arrays, such as the `resultSet` array returned from a Search action (as in the previous example), have column names saved in row zero of the array.

As in the previous example, you can use the row zero column names to access the columns in the array using array-returning syntax. You can also return these values (column names) by specifying row and column co-ordinates `[0,*]`. For example, to return only column names from a `resultSet` array:

```
<@VAR NAME="resultSet[0,*]">
```

When you use Meta Tags other than <@VAR> which return arrays (such as <@DATASOURCESTATUS>), you cannot access row zero of that array directly, even though it exists. To access row zero, you must put the returned result of the array-returning Meta Tag into a temporary array, and then return row zero of that temporary array.

For example:

For more information, see "<@DATASOURCE-STATUS>" on page 77.

```
<@ASSIGN NAME=tempArray SCOPE=local
value=<@DATASOURCESTATUS>>

<@VAR NAME=tempArray[0,*]>
```

How TeraScript Determines Default Scope in Variable Assignments

When you assign a value to a variable but do not specify a scope, TeraScript performs these steps to determine which scope to use:

- TeraScript looks for an existing variable with that name. The search starts in the **request** scope, and continues up through **user**, **cookie**, **application**, **domain**, and finally to **system**

scope. If TeraScript finds the variable, it assigns the value to it and stops looking.

Caution TeraScript does not search any custom scope. If you want to assign a value to or retrieve a value from a custom scope, you must explicitly specify the scope.

For more information, see defaultScope on page 409.

- If no variable with the specified name is found, TeraScript creates the variable in the default scope for new variables. This is by default request scope, and can be changed using the `defaultScope` configuration variable.

Here are some examples. Assume that these variables are already defined:

Name	Scope
foo	request
doh	domain
ipsum	user
lorem	domain

`<@ASSIGN NAME="foo" VALUE="myVal">` assigns `myVal` to the existing local variable `foo`.

`<@ASSIGN NAME="doh" VALUE="myVal">` creates a new request variable called `doh` and assigns `myVal` to it.

`<@ASSIGN NAME="ipsum" VALUE="myVal" SCOPE="request">` creates a new request variable called `ipsum` and assigns `myVal` to it.

`<@ASSIGN NAME="lorem" VALUE="myVal">` assigns `myVal` to the domain variable `lorem`.

Using Configuration Variables

Configuration variables are special values that control basic TeraScript behaviors. For example, there are configuration variables for controlling such settings as:

- the type of information written to TeraScript Server's log file (`loggingLevel`)
- the default date format used by TeraScript Server (`dateFormat`)
- how long user variables last before expiring (`variableTimeout`).

For a complete and detailed list of configuration variables, see Configuration Variables on page 373.

Some configuration variables can be set in all scopes (except cookie scope), some in particular scopes, and some only in system scope. For those configuration variables that can be set in all scopes, the different scopes have the following effects:

- **scope**=**SYSTEM** affects *all* application files executed by TeraScript Server. Assignments made to these configuration variables remain in effect until you change them again (even after stopping and starting TeraScript Server). The values of these variables are saved in the `t4server.ini` configuration file.

  The TeraScript Administration Application (the `config.taf` application file) provided on the TeraScript website makes it easy to change the values of these configuration variables from your Web browser. You need to know the password set by `configPasswd` in order to set configuration variables for the system.

  For example, the configuration variable `dateFormat scope=SYSTEM` would set the format of the date returned by such Meta Tags as `<@CURRENTDATE>` in all TeraScript application files served by TeraScript Server.

- **scope**=**DOMAIN** affects the configuration variables in a particular TeraScript domain.

  For example, the configuration variable `dateFormat scope=DOMAIN` would set the format of the date returned by such Meta Tags as `<@CURRENTDATE>` in all TeraScript application files served in a particular TeraScript domain.

- **scope**=**APPLICATION** affects application files within a particular TeraScript application, as defined in the application configuration file.

For example, there are situations where it is useful to have a different `dateFormat` in a different application. A company that does business in French and in English is being hosted on the same Web server, in different TeraScript applications. Because of the different conventions for date formats in the different languages, they would want the date to look quite different in each application—English or French.

The configuration variable `dateFormat` with `scope=APPLICATION` would be set to different values within each application, and from then on, dates returned by TeraScript (with a Meta Tag such as `<@CURRENTDATE>`) would have different formats in the different applications.

For more information, see userKey, altuserKey on page 474.

- **scope=USER** affects application files executed by a particular user. As with normal user variables, these depend on the setting of a reliable user key in order to work as expected, and they expire 30 minutes after that user last accesses TeraScript.

  For example, you could offer the user the chance to set the `dateFormat scope=USER` variable, and, for that particular user from that point on, dates would be formatted the way that user wants.

- **scope=REQUEST** affects a particular TeraScript application file execution, to override the system, domain, or user settings in a particular case. The change is effective from the action in which you make the assignment until the end of the application file's execution (including all its branches if it branches to another application file, and all methods called). For example, you can change the `dateFormat` for a particular application file.

  Any configuration variables that are valid in request scope are also valid in the instance and method scope.

You assign values to configuration variables in the same way as assignments to other kinds of variables: by using TeraScribe's Assign action or by using the `<@ASSIGN>` Meta Tag to set configuration variables in HTML processed by TeraScript.

Caution Users cannot set system-level configuration variables unless they know the administrative password.

Using User Keys

To associate a user variable with a particular user, TeraScript must have a piece of information that it can use to uniquely identify that user.

TeraScript refers to the unique identifier used for tracking a user's variables as the `user key`. TeraScript has several settings allowing you to control what information is used as the user key. TeraScript default behavior is to use three Meta Tags as the value of `userKey`:

```
<@APPKEY><@USERREFERENCE><@CGIPARAM CLIENT_IP>
```

These parts of the `userKey` function as follows:

For more information, see <@APPKEY> on page 24, <@CGIPARAM> on page 57, and <@USERREFERENCE> on page 283.

- `<@APPKEY>` returns the key value of the current application scope. This ensures that users in different applications cannot share variables.
- `<@USERREFERENCE>` generates a unique number for tracking each user.
- `<@CGIPARAM CLIENT_IP>` returns the IP address of the user who is accessing a particular TeraScript application file. This ensures that a session can not be taken over by someone from another IP address.

Under this scheme, when users execute their first TeraScript application file, TeraScript generates a unique user reference number and uses it as the user key. In the results sent back to the user, TeraScript includes the user reference number as an HTTP cookie. This cookie is remembered by the Web browser and is sent automatically with every subsequent request to your server. TeraScript checks for the existence of the cookie whenever it accesses a user variable. If it was sent, the cookie value is used as the user key.

To help understand how user variable tracking works, imagine that two users, John and Simone, have each executed an application file that assigns a value to a user variable called favorite_color. TeraScript generates a unique user reference number for each user and sends it in a cookie back to their Web browsers. The user reference number is a 24-digit hexadecimal string. Inside TeraScript

Server, the user variable information is organized in a manner similar to this:

User	User Key Value(<@APPKEY> <@USERREFERENCE> <@CGIPARAM CLIENT_IP>)	Variable Name	Var Value
John	7F00000146B4488D0C5B847CA5853794E38C12.21.21.212	favorite_color	blue
Simone	54A497684AD2A5853794E38C5940014FDD1316.01.27.128	favorite_color	red

When, in another application file, the user variable `favorite_color` is referenced, TeraScript first checks to see what the value of the user key is for the current user. It then uses that key value in combination with the user variable name to determine the value to return. If the user is John, the user key value, sent to John's Web browser as a cookie, is `7F00000146B4488D0C5B847CA5853794E38C12.21.21.212`, and the user variable reference returns "blue"; if the user is Simone, the user key value is `54A497684AD2A5853794E38C5940014FDD1316.01.27.128` and it returns "red".

User Keys Specific to Transactions

In the results sent back to the user, TeraScript includes the user reference number as an HTTP cookie. This cookie is remembered by the Web browser and is sent automatically with every subsequent request to your server. Cookies are common to the Web browser application, not specific to a Web browser window. If a user opens two windows in a Web browser application, both windows share the same cookies and, therefore, the same user variables. Usually, this is what you want.

Sometimes you want the user variables to be specific to a particular transaction. In this case, you should store the needed values not as user variables, but as hidden form fields or search arguments.

Not all Web browser applications support cookies. Currently, the two major cookie-capable Web browsers are Netscape Navigator and Microsoft Internet Explorer. If you need to support user reference-based user variables with Web browsers that do not support cookies, TeraScript allows the user reference number to be passed via a special search argument, `_userReference`. The search argument *must* be passed with every URL.

For example:

```
<A HREF="<@CGI>/purchase_item.tafitem_num=
<@COLUMN item_num>&_userReference=
<@USERREFERENCE>">Process Order</A>
```

There is also a shortcut Meta Tag for including the entire search argument, <@USERREFERENCEARGUMENT>.

Changing the User Key

TeraScript gives you full control over what information is used as the user variable key. It does this via two configuration variables: `userKey` and `altUserKey`.

The contents of `userKey` determines the default key used for tracking user variables. The contents of `altUserKey` with `SCOPE=system` or `domain` determines what key is used when the value of the key specified by `userKey` with `SCOPE=system` or `domain` evaluates to empty. As stated above, the default value for `userKey` is `<@APPKEY><@USERREFERENCE><@CGI CLIENT_IP>`. The default value for `altUserKey` is empty.

Note The `userKey` and `altUserKey` specified in the system scope are the default keys. If the domain scope `userKey` and `altUserKey` have been set, their values override the system settings and determine the user key for users in that domain, because of the way that variables are evaluated in TeraScript.

Assigning Values to userKey and altUserKey

You can assign value to `userKey` and `altUserKey` using the TeraScript Configuration Manager (the `config.taf` application file).

When you assign a value to `userKey` and `altUserKey`, you must tell TeraScript Server not to evaluate Meta Tags in the `VALUE` attribute, but instead to evaluate the Meta Tag when user variables need to be keyed. This is done with the `<@LITERAL>` Meta Tag.

For more information, see <@LITERAL> on page 195, <@ASSIGN> on page 32, and "<@USERREFERENCE>" on page 283

The syntax of the assignment to `userKey` of its default value would be as follows:

```
<@ASSIGN NAME="userKey" VALUE="<@LITERAL
VALUE='<@APPKEY><@USERREFERENCE><@CGIPARAM
CLIENT_IP>'">
```

Alternate User Keys

Here are some alternate possibilities for `userKey` (and `altUserKey`). You must use the `<@LITERAL>` tag when assigning to these configuration variables:

See "<@CGIPARAM>" on page 57.

- `<@CGIPARAM USERNAME>`

If you are using HTTP authentication for your site or for a particular set of TeraScript application files, and have each user logging in with a unique user name, this user name can be used to identify users and their user variables.

- `<@CGIPARAM CLIENT_IP>`

  TeraScript can use the client's IP address as a user key. This user keying mechanism is useful when you know that the users hitting your site are guaranteed to have a one-to-one user/IP address mapping.

  Unfortunately, in many situations, the IP address is not an accurate method of identifying a particular user. For instance, some corporate networks are set up so that all HTTP requests are routed through a single server. In this case, requests from different users may all have the same IP address. When user variables are keyed on IP addresses and an address may represent several users, user variables do not serve their purpose of providing a way to keep user-specific data.

Returning the Value of userKey and altUserKey

When the `userKey` and `altUserKey` configuration variables are used in Results HTML, they evaluate to the text of the tags, not the tag values. To see the current value of the user key, use the `ENCODING=METAHTML` parameter to the `<@VAR>` Meta Tag. For example, if the following text is typed in a Results HTML field:

```
Variables are now being keyed on:
<@VAR NAME="userKey" scope=SYSTEM>.
⇒ TeraScript returns <@APPKEY><@USERREFERENCE><@CGI
CLIENT_IP>

The value of the key in the current execution is:
<@VAR NAME="userKey" scope="SYSTEM"
 ENCODING="METAHTML">
⇒ TeraScript returns
7F00000146B4488D0C5B847CA5853794E38C
```

Using Application File User Keys

You can override the default user key on an application file basis by setting `userKey` and `altUserKey` with `scope=LOCAL`. These work just like their system-wide counterparts, but apply only until the end of the application file execution. Use local user keys when you want to temporarily use a user key different than the system user key.

CHAPTER 5

Document Object Model

5

Creating and Manipulating Document Instances Using DOM

This chapter describes the Document Object Model (DOM), which allows users to manipulate XML structures, and shows how it is used in TeraScript to create, manipulate, and return the values of document instances.

The topics covered in this chapter include:

- definition of DOM
- overview of using DOM
- XPointer syntax
- XPath syntax
- manipulating a document instance
- returning XML
- applications of DOM
    - building complex data structures
    - separating business and presentation logic
    - working with TeraScript application files.

What is DOM?

DOM is the *Document Object Model*, a World Wide Web Consortium (W3C) standard for the manipulation of structured data, including XML.

For more information on the Document Object Model, see www.w3.org/DOM/.

DOM, as the name implies, allows TeraScript developers to manipulate the elements of a structured document (for example, XML) as if they were objects. Developers can build document instances, navigate their structure, and add, modify, or delete elements and content. DOM creates a representation of an XML document that is an *object tree*, and gives you the tools to create and manipulate the object tree in TeraScript using TeraScript variables and Meta Tags.

TeraScript adds another intrinsic data type for TeraScript variables (in addition to strings and arrays): *document instance*, which is an XML document represented using DOM. Once an XML document has been converted to a document instance, you can manipulate the document instance using TeraScript Meta Tags.

DOM allows you to do the following:

- Create intermediate complex data structures in XML format. For more information, see Creating Complex Data Structures on page 371.
- Consolidate business logic (for example, database searches and the building up of retrieved results) separately from presentation logic (for example, HTML pages sent to a Web server). For more information, See "Separating Business and Presentation Logic" on page 373.
- Create, manipulate, read in, and write out TeraScript application files (which are in XML format). For more information, See "Reading and Writing TeraScript Application Files" on page 375.
- Create and manipulate other XML structures for a variety of purposes (for example, Electronic Data Interchange [EDI]).

Caution Not all features of XML are accessible through DOM. Elements, attributes, and element contents are accessible; inaccessible XML features include those that are not useful for presentation, for example, comments and processing instructions.

The following sections of this chapter describe the syntax and Meta Tags for manipulating a document instance, and show some applications of DOM.

Overview of Using DOM

The following steps give an overview of using DOM to create and manipulate XML in TeraScript:

1. Set up a document instance in a TeraScript variable by doing one of the following:
   - Create a document instance in a TeraScript variable using <@DOMINSERT>.
   - Explicitly assign XML to a TeraScript variable using the <@ASSIGN> and <@DOM> Meta Tags or the Assign action.
   - Use the File Read action or <@INCLUDE> Meta Tag to assign an XML document to a TeraScript variable, using <@DOM> or <@DOMINSERT>.
2. Use DOM Meta Tags to manipulate the document instance in a TeraScript variable.

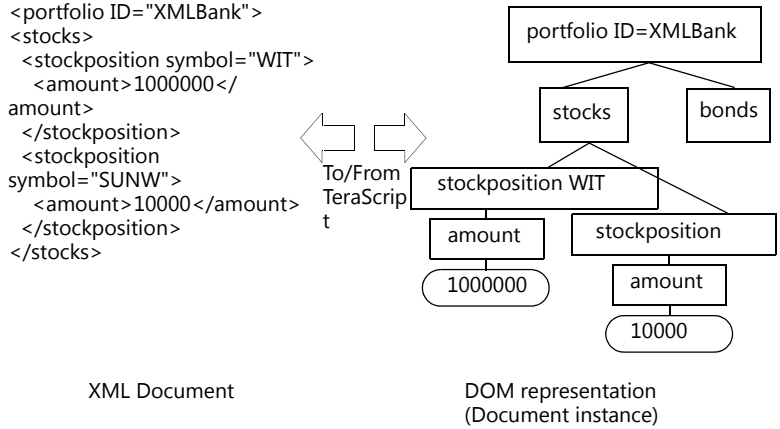
   The <@DOMINSERT>, <@DOMDELETE>, and <@DOMREPLACE> Meta Tags manipulate the document instance. *XPointer syntax* is used to select an element or groups of elements to be manipulated.
3. Return values from the document instance into your application.

   You can return the entire XML that you have manipulated by returning the value of the document instance TeraScript variable.

   You can return individual element names, values, attribute, or range of attributes within the document instance by using the <@VAR>, <@ELEMENTNAME>, <@ELEMENTVALUE>, <@ELEMENTATTRIBUTE>, and <@ELEMENTATTRIBUTES> Meta Tags. These Meta Tags use XPointer syntax.

Example

The following diagram shows a schematic representation of an XML document and its DOM representation.



To manipulate XML in TeraScript, the XML must be converted to a DOM representation. This is generally done by using the <@ASSIGN> Meta Tag or Assign action in conjunction with the <@DOM> Meta Tag or with the use of the <@DOMINSERT> tag. The following creates a document instance in the myDom variable in request scope:

```
<@ASSIGN NAME="myDom" SCOPE="request"
VALUE="<@DOM VALUE='<portfolio ID="XMLBank">
<stocks>
  <stockposition symbol="WIT">
    <amount>1000000</amount>
  </stockposition>
  <stockposition symbol="SUNW">
    <amount>10000</amount>
  </stockposition>
</stocks>
<bonds/>
</portfolio>'>">
```

The XML document instance can then be manipulated by TeraScript Meta Tags. For example, using the above representation, the following Meta Tag deletes all <STOCKPOSITION> elements that have the attribute name SYMBOL with the attribute value SUNW (in this case, there is only one):

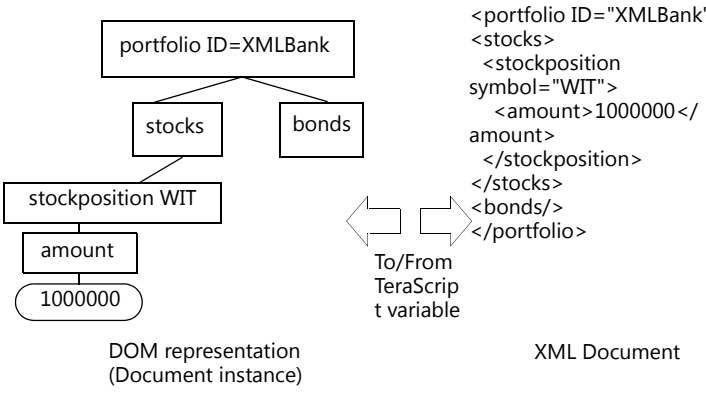
XPointer:

```
<@DOMDELETE OBJECT="request$myDom"
XPOINTER="root().descendant(all,stockposition,symbo
l,SUNW)">
```

XPath:

```
<@DOMDELETE OBJECT="request$myDom" XPATH="//
stockposition[@symbol='SUNW']">
```

All sub-elements and data of this element are also deleted. When the XML is returned within a TeraScript application (for example, with the <@VAR> Meta Tag), the document instance is saved out as XML.



XPointer Syntax

The syntax is a stripped-down version of that proposed by a W3C Working Group. See www.w3.org/TR/WD-xptr

XPointer syntax is used by the various <@DOM...> and <@ELEMENT...> Meta Tags to point at one or more elements in the document instance. In general, an XPointer is a series of terms, linked together by a period, that navigate to a particular element (or elements). For example:

```
root().child(1).child(2)
```

The pointer is a mixture of *absolute* and *relative* terms, where absolute terms refer to a specific element, and relative terms refer to an element by way of its relation to another element. The following pointer terms are implemented in TeraScript :

- **root**()
- **id**(*idvalue*)
- **child**(*number* or *all*,*nodetype*,*attribname*,*attribvalue*)
- **descendant**(*number* or *all*,*nodetype*,*attribname*,*attribvalue*)

Additionally, for the child and descendant terms, only the `#element` node type, and specific element names, are supported.

Root

If an XPointer begins with `root()`, the location source is the root element of the containing resource. If an XPointer omits any leading absolute location term, it is assumed to have a leading `root()` absolute location term.

ID

If an XPointer term is of the form `id(Name)`, the location source (the point from which that XPointer starts in the DOM tree) is the element in the containing resource (in this case, all or part of the DOM tree) with an attribute having a declared type of `id` and a value matching the given `Name`.

For example, the location term `id(a27)` chooses the necessarily unique element of the containing resource which has an attribute declared to be of type `id` whose value is `a27`.

Child

Identifies direct child nodes of the location source. Child nodes are nodes that are exactly one step downwards from a node.

Descendant

The `descendant` keyword selects a node of the specified type anywhere inside the location source, either directly or indirectly nested.

The descendant location term looks down through trees of subelements in order to end at the node type requested. The search for matching node types occurs in the same order that the start-tags of elements occur in the XML data stream: the first child of the location source is tested first, then (if it is an element) that element's first child, and so on. In formal terms, this is a depth-first traversal.

Terms of Child or Descendant

The following terms can be used with the child and descendant keywords:

- number or all

  For a positive number *n*, the *n*th of the candidate locations is identified. If the instance value all is given, then all the candidate locations are selected. The following example identifies the fifth child element:

  ```
  child(5)
  ```

- nodetype

  The node type may be specified by one of the following values:

  - Name

    Selects a particular XML element type; only elements of the specified type will count as candidates. For example, the following identifies the 29th paragraph of the fourth sub-division of the third major division of the location source:

    ```
    child(3,DIV1).child(4,DIV2).child(29,P)
    ```

  - #element

    Identifies XML elements. If no Name is specified, #element is the default.

- attribname, attribvalue

  The attribname and attribvalue arguments are used to provide attribute names and values to use in selecting among candidate elements. The attribvalue argument is always case-sensitive.

  Attribute names may be specified as "*" in location terms in the (unlikely) event that an attribute value constitutes a constraint regardless of what attribute name it is a value for.

  For example, the following location term selects the first child of the location source for which the attribute TARGET has a value:

```
child(1,#element,TARGET,*)
```

The following XPointer chooses an element using the N attribute:

```
child(1,#element,N,2).(1,#element,N,1)
```

Beginning at the location source, the first child (whatever element type it is) with an `N` attribute having the value `2` is chosen; then that element's first child element having the value `1` for the same attribute is chosen.

A child with an invalid specification (for example, out-of-range number) returns an error from TeraScript Server.

Example

Given the following document instance:

```
<portfolio ID="XMLBank">
       <stocks>
              <stockposition symbol="WIT">
                     <amount>1000000</amount>
              </stockposition>
              <stockposition symbol="MSFT">
                     <amount>200000</amount>
              </stockposition>
              <stockposition symbol="SUNW">
                     <amount>10000</amount>
              </stockposition>
       </stocks>
       <bonds/>
       <cash>
              <cad/>
              <usd>
                     <amount>100000</amount>
              </usd>
       </cash>
</portfolio>
```

- `root()` returns the `portfolio` element, as would `id(XMLBank)`.
- `root().child(1).child(2)` returns the second `stockposition` element (`symbol` attribute and `MSFT` attribute value), as does `root().child(1).child(1,*,symbol,MSFT)`.
- `descendant(3,amount)` goes right to the third amount element in the tree, without having to specify its parentage. `descendant(all,stockposition)` returns a list of the three stockposition elements.

- `child(4)` returns nothing, because, while an omitted initial absolute term always implicitly adds `root()`, there is no fourth child of the `portfolio` element.

XPath Syntax

XPath is used by the various <@DOM...> and <@ELEMENT...> , Meta Tags 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()

Node-Set
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

Manipulating a Document Instance

This section gives details on creating and manipulating a document instance using DOM Meta Tags.

Creating a Document Instance

The first step in manipulating a document instance is to create one from the XML.

To create a document instance

Do one of the following:

- Use the <@DOM> Meta Tag in conjunction with an Assign action or the <@ASSIGN> Meta Tag.
- Use the <@DOMINSERT> Meta Tag, specifying a new variable (and optionally a scope specification). The XML to be inserted is found between the start- and end-tags.

The following examples create a document instance in the variable named myDom in application scope. If that variable already exists in that scope, the value of that variable is replaced:

```
<@ASSIGN NAME="myDom" SCOPE=application
VALUE="<@DOM VALUE='<XML><DIV ID="1"><P>This is
an example of a structured document.</P></DIV><DIV
ID="2"><P>Here is some more text.</P><P>Here is an
additional paragraph of text.</P></DIV></XML>'>">
```

```
<@DOMINSERT OBJECT="myDom" SCOPE=application>
<XML>
<DIV ID="1">
<P>This is an example of a structured document.</P>
</DIV>
<DIV ID="2">
<P>Here is some more text.</P>
<P>Here is an additional paragraph of text.</P>
</DIV>
</XML>
</@DOMINSERT>
```

There are also several other different ways you can create a document instance from XML, but they all involve variations on the basic use of <@DOM> and <@ASSIGN>, or <@DOMINSERT>. For example, you can read in an XML file using <@INCLUDE>, and create a document instance with <@DOM> or <@DOMINSERT>:

```
<@ASSIGN NAME=myXML VALUE=<@DOM VALUE=<@INCLUDE
FILE=fred.xml>>>
```

```
<@DOMINSERT OBJECT=myXML>
<@INCLUDE FILE=fred.xml>
</@DOMINSERT>
```

Using DOM Meta Tags

You manipulate the XML document by using the DOM Meta Tags to point to the element(s) you want to delete, replace, or insert into. For inserting and replacing Meta Tags, the XML to be inserted or replaced is found between start- and end-tags of the DOM Meta Tag.

To insert XML into a document instance

- Use the <@DOMINSERT> Meta Tag.

For example, using the above myDom document instance, the following inserts an additional paragraph between the two <P> elements in the second <DIV> element:

Using XPointer Syntax:

```
<@DOMINSERT OBJECT="myDom" SCOPE="application"
XPOINTER="root().child(2).child(1)"
POSITION="BEFORE">
<P>Here is an additional paragraph of text.</P>
</@DOMINSERT>
```

Using XPath Syntax:

```
<@DOMINSERT OBJECT="myDom" SCOPE="application"
XPATH="/XML/DIV[position()=2]/P[position()=1]"
POSITION="BEFORE">

<P>Here is an additional paragraph of text.</P>

</@DOMINSERT>
```

To delete XML from a document instance

- Use the <@DOMDELETE> Meta Tag.

For example, using the above myDom document instance, the following deletes the second paragraph in the second <DIV> element.

Using XPointer Syntax:

```
<@DOMDELETE OBJECT="myDom" SCOPE="application"
XPOINTER="root().child(2).child(2)">
```

Using XPath Syntax:

```
<@DOMDELETE OBJECT="myDom" SCOPE="application"
XPATH="/XML/DIV[position()=2]/P[position()=2]">
```

To replace XML in a document instance

- Use the <@DOMREPLACE> Meta Tag.

For example, using the above myDom document instance, the following replaces the first <DIV> element (the one with the attribute ID=1).

Using XPointer Syntax:

```
<@DOMREPLACE OBJECT="myDom" SCOPE="application"
XPOINTER="root().descendant(all,DIV,ID,1)">
<DIV ID="1"><P>Here is a replacement paragraph
inside a replacement DIV.</P>
</DIV>
</@DOMREPLACE>
```

Using XPath Syntax:

```
<@DOMREPLACE OBJECT="myDom" SCOPE="application"
XPATH="//DIV[@ID='1']">

<DIV ID="1"><P>Here is a replacement paragraph
inside a replacement DIV.</P>

</DIV>

</@DOMREPLACE>
```

Returning XML in TeraScript Applications

You can use a variety of TeraScript Meta Tags to return XML from a document instance. You can return all or part of the document instance using <@VAR>, return particular element names with <@ELEMENTNAME>, return element values with <@ELEMENTVALUE>, return attribute values with <@ELEMENTATTRIBUTE>, or return all attributes of one or more elements with <@ELEMENTATTRIBUTES>.

When returning values, you can select whether you want to return the value as text or as an array. By default, multiple values returned by the <@ELEMENT...> Meta Tags are returned as an array, and single values are returned as text.

If the XPOINTER/XPATH attribute of any of the XML-returning Meta Tags is omitted, the root element of the document instance is assumed.

The following examples assume there is a document instance variable called myDom which contains a DOM representation of the following XML:

```
<XML>
<DIV ID="1" CLASS="normal">
<P>This is an example of a structured document.</P>
</DIV>
<DIV ID="2" CLASS="urgent">
<P>Here is some more text.</P>
<P>Here is an additional paragraph of text.</P>
</DIV>
</XML>
```

Using <@VAR> and <@ASSIGN> With DOM

To return the entire or part of a document instance

- Use the <@VAR> Meta Tag.

The following example returns the entire document instance:

```
<@VAR NAME="myDom">
```

A document instance is returned by <@VAR> as XML with no conversion of characters to HTML entities if the ENCODING attribute is not present. No conversion occurs even when XML is placed in HTML (for example, to be displayed as a Web page). All other ENCODING attribute settings function normally.

Note The use of <@VAR> with XML when no ENCODING attribute is specified differs from returning other types of variables—text and arrays—into HTML. The default behavior of <@VAR> is to return variables as encoded for HTML, so that the returned value displays literally in the HTML (for example, "<" and ">" characters are encoded as their HTML entity definitions: < and >). The lack of encoding when returning document instances reflects the fact that XML is normally intended as instructions to the client (like HTML), generally not as data to be displayed.

In order to display the XML in its encoded form—for example, for display in a Web browser—you can use the ENCODING=MULTILINEHTML attribute when using <@VAR>, which converts the appropriate text to HTML entities but also adds a
 HTML tag to the end of each line. You can also display encoded XML by first assigning the XML text to a variable (using the TYPE=TEXT attribute, which forces XML to be returned, not a document instance), and then returning the value of that variable in the HTML. The default encoding of variables returned with <@VAR> then takes place, for example:

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

The XML is returned with the appropriate text converted to HTML entities.

If you want to return part of the document instance, you can do so by using the XPOINTER/XPATH attribute of the <@VAR> Meta Tag. This is a pointer to the element that you want to return. All sub-elements and values within that sub-element are returned as well.

For example, the following returns the second <DIV> element from the above document instance, all sub-elements, and data in those elements:

XPointer:

```
<@VAR NAME="myDom" XPOINTER="root().child(2)">
```

XPath:

```
<@VAR NAME="myDom" XPATH="/XML/DIV[position()=2]">
```

returns:

```
<DIV CLASS="urgent" ID="2">
<P>Here is some more text.</P>
<P>Here is an additional paragraph of text.</P>
</DIV>
```

Copying all or part of a document instance to another

variable

- Use the <@VAR> Meta Tag in conjunction with the <@ASSIGN> Meta Tag.

There are two possible cases: where the XPOINTER/XPATH attribute of <@VAR> points at a single element or at multiple elements:

- When the XPOINTER/XPATH attribute of <@VAR> points at a *single* element, the element and its children are copied into the variable defined by the <@ASSIGN> Meta Tag.

  The following example assigns the second <DIV> element from the above document instance, all sub-elements, and data in those elements to a new variable in user scope called newDom:

XPointer:

```
<@ASSIGN NAME="newDom" SCOPE="user" VALUE="<@VAR
NAME='MyDom' XPOINTER='root().child(2)'>">
```

XPath:

```
<@ASSIGN NAME="newDom" SCOPE="user" VALUE="<@VAR
NAME='MyDom' XPATH='//*[position()=2]'>">
```

- When the XPOINTER/XPATH attribute of <@VAR> points at *multiple* elements, the elements and their children are copied into the variable defined by the <@ASSIGN> Meta Tag; however, because a document must have a single root element, a root element called <root> is automatically created as the parent of the copied elements and the root of the document instance.

  The following example assigns all the <P> elements in the myDom variable to a new variable in request scope called PDOM:

XPointer:

```
<@ASSIGN NAME="PDOM" SCOPE="request" VALUE="<@VAR
NAME='MyDom' XPOINTER='descendant(all,P)'>">
```

XPath:

```
<@ASSIGN NAME="PDOM" SCOPE="request" VALUE="<@VAR
NAME='MyDom' XPATH='//P'>">
```

This results in the following document instance in the PDOM variable:

```
<root>
<P>This is an example of a structured document.
</P>
```

```
<P>Here is some more text.</P>
<P>Here is an additional paragraph of text.</P>
</root>
```

Using <@ELEMENT...> Meta Tags With DOM

All of the <@ELEMENT...> Meta Tags can return either the text representation of an array (TYPE attribute set to TEXT), or an actual array (TYPE attribute set to ARRAY). However, when an array is returned in Results HTML, it is always returned as an HTML table, whether the TYPE attribute of the <@ELEMENT...> Meta Tags is set to TEXT or ARRAY.

For more information, see Arrays on page 330.

When an array is referenced within a variable assignment, setting the TYPE attribute to TEXT returns the HTML table; setting the type attribute to ARRAY or not putting the attribute in the Meta Tag (the default) copies the array or part of the array, and the array is not converted to an HTML representation.

To return one or more element names

- Use the <@ELEMENTNAME> Meta Tag.

For more information, see <@ELEMENTNAME> on page 129 .

For example, the following returns the name of the element that is being pointed to from the above document instance:

XPointer:

```
<@ELEMENTNAME OBJECT="myDom"
XPOINTER="root().child(2)">
```

XPath:

```
<@ELEMENTNAME OBJECT="myDom" XPATH="/XML/
DIV[position()=2]">
```

Returns:

```
DIV
```

The following example returns two element names as an array:

XPointer:

```
<@ELEMENTNAME OBJECT="myDom"
XPOINTER="root().child(all)">
```

XPath:

```
<@ELEMENTNAME OBJECT="myDom" XPATH="/XML/*">
```

Returns:

```
DIV
```

DIV

To return one or more attribute values

- Use the <@ELEMENTATTRIBUTE> Meta Tag.

For more information, see <@ELEMENTATTRIBUTE> on page 124 .

For example, the following returns the value of the attribute named ID in the element that is being pointed to from the above document instance:

XPointer:

```
<@ELEMENTATTRIBUTE OBJECT="myDom" ATTRIBUTE="ID"
XPOINTER="root().child(2)">
```

XPath:

```
<@ELEMENTATTRIBUTE OBJECT="myDom" ATTRIBUTE="ID"
XPATH="/XML/DIV[position()=2]">
```

Returns:

```
2
```

The following example returns two attribute values as an array:

XPointer:

```
<@ELEMENTATTRIBUTE OBJECT="myDom" ATTRIBUTE="ID"
XPOINTER="root().child(all)">
```

XPath:

```
<@ELEMENTATTRIBUTE OBJECT="myDom" ATTRIBUTE="ID"
XPATH="/XML/*">
```

Returns:

1
2

To return all attribute values of an element or elements

- Use the <@ELEMENTATTRIBUTES> Meta Tag.

For more information, see <@ELEMENTATTRIBUTES> on page 127 .

This Meta Tag returns a one-dimensional array if you are pointing at one element, and a two-dimensional array if you are pointing at multiple elements.

For example, the following returns the attribute names and values of the elements that are being pointed to from the above document instance. Each element pointed to returns a row. The returned value is a two-dimensional array:

XPointer:

```
<@ELEMENTATTRIBUTES OBJECT="myDom"
XPOINTER="root().child(all)">
```

XPath:

```
<@ELEMENTATTRIBUTES OBJECT="myDom" XPATH="/XML/*">
```

Returns:

normal	1
urgent	2

Row 0 (zero) of the array contains the attribute name for each column (in this case, `ID` and `CLASS`, respectively).

To return one or more element values

- Use the <@ELEMENTVALUE> Meta Tag.

For more information, see <@ELEMENTVALUE> on page 132 .

For example, the following returns the value of the element that is being pointed to from the above document instance:

XPointer:

```
<@ELEMENTVALUE OBJECT="myDom"
XPOINTER="root().child(1).child(1)">
```

XPath:

```
<@ELEMENTVALUE OBJECT="myDom" XPATH="/XML/
DIV[position()=1]/P[position()=1]">
```

Returns:

```
This is an example of a structured document.
```

The following example returns the element values of the elements that are being pointed to from the above document instance. The returned value is a one-dimensional array:

XPointer:

```
<@ELEMENTVALUE OBJECT="myDom"
XPOINTER="root().child(2).child(all)">
```

XPath:

```
<@ELEMENTVALUE OBJECT="myDom" XPATH="/XML/
DIV[position()=2]/*">
```

Returns:

Here is some more text.
Here is an additional paragraph of text.

Applications of DOM

DOM allows you to parse XML. Many different standards are written in XML, and the ability of TeraScript to manipulate structured data enables you to read, modify, and write XML for a variety of purposes.

This section discusses some of the uses of the Document Object Model, including the building up of complex data structures in application files, the separation of presentation and business logic, and reading and writing TeraScript application files (which are in XML format).

Creating Complex Data Structures

Using DOM, you can build up complex data structures in document instances with data drawn from a variety of sources. This data can then be returned to the user using the DOM Meta Tags as XML, HTML, or text, in a variety of complex ways.

The steps to creating and using complex data structures are:

1. Decide on a structure for the data.

   In order for a complex structure to be created, there has to be agreement on some sort of document type definition. This may not be formally specified as a DTD, but the general tree structure, elements, attributes, and their ordering should be clear.

2. Get the data from whatever source (data sources, external actions, objects, and so on).

3. Use TeraScript variables to save the intermediate results. Generally, results from an action are returned with the `resultSet` array. Use this array or portions of this array to create different TeraScript variables or arrays that save the relevant information from a database action.

4. Incorporate the data into the structure of the document instance by using `<@DOM...>` Meta Tags to insert and modify XML.

5. Return values from the document instance using `<@ELEMENT...>` Meta Tags.

For more information, see Using DOM Meta Tags on page 363.

For more information, see Returning XML in TeraScript Applications on page 365.

Example

Creation of a complex user portfolio of stocks, bonds, cash, and other financial assets may require a variety of searches into various financial companies' data sources, using data returned from an object that retrieves financial data, using external actions to return data from a script that reads a custom financial format, and so on. Creating a document type definition and using document instances is a way to organize all this complex data so that it can be returned using DOM Meta Tags.

The tree structure of a portfolio could look something like the following:

```
<portfolio ID="">
       <stocks>
              <stockposition symbol=""></stockposition>
       </stocks>
       <bonds>
       </bonds>
       <mutualfunds>
              <mfposition symbol=""></mfposition>
       </mutualfunds>
       <cash>
              <cad></cad>
              <usd></usd>
       </cash>
</portfolio>
```

While the stocks and bond data could be retrieved from two different data sources, the cash and mutual funds data require, respectively, a call to an object and one to an external script. The data returned from all these different types could be easily inserted into the data structure with the use of DOM Meta Tags.

Assuming that the results from a search on stocks held by a particular user were returned and copied from the `resultSet` of that action into a `mystocks` array, the following Meta Tags return values by looping through an array where the stock symbol is in column one, and the number of shares being held is in column two, and inserts those values into a `stock` and `amount` element within the `Portfolio` document instance:

Using XPointer Syntax:

```
<@DOMINSERT OBJECT="Portfolio"
XPOINTER="root().child(1,stocks)"><@FOR
STOP="<@NUMROWS ARRAY='mystocks'>"><stockposition
symbol="<@VAR
```

```
NAME='mystocks[<@CURROW>,1]'>"><amount><@VAR
NAME="mystocks[<@CURROW>,2]"></amount></
stockposition></@FOR></@DOMINSERT>
```

Using XPath Syntax:

```
<@DOMINSERT OBJECT="Portfolio" XPATH="/portfolio/
stocks"> <@FOR STOP="<@NUMROWS
ARRAY='mystocks'>"><stockposition symbol="<@VAR
NAME='mystocks[<@CURROW>,1]'>"><amount><@VAR
NAME="mystocks[<@CURROW>,2]"></amount></
stockposition></@FOR></@DOMINSERT>
```

To return values from the document instance, use the ELEMENT Meta Tags which refer to various element or attribute values in the document instance.

For example, to return an array of all the stock symbols in the portfolio, use the following Meta Tag:

Using XPointer Syntax:

```
<@ELEMENTATTRIBUTE OBJECT="Portfolio"
ELEMENT="root().descendant(all)"
ATTRIBUTE="symbol">
```

Using XPath Syntax:

```
<@ELEMENTATTRIBUTE OBJECT="Portfolio" XPATH="//*"
ATTRIBUTE="symbol">
```

Separating Business and Presentation Logic

TeraScript's action based metaphor allows you to build up returned HTML on Web pages, action by action, in a variety of complex ways. This model is a good one for many Web applications. However, it may sometimes be useful in a TeraScript solution to think about separating business and presentation logic.

The *business logic* of a TeraScript application is the various actions and calls that retrieve information. The *presentation logic* of a TeraScript application is how Web pages are shown to the user; that is, the path that the user goes through as they use a Web site.

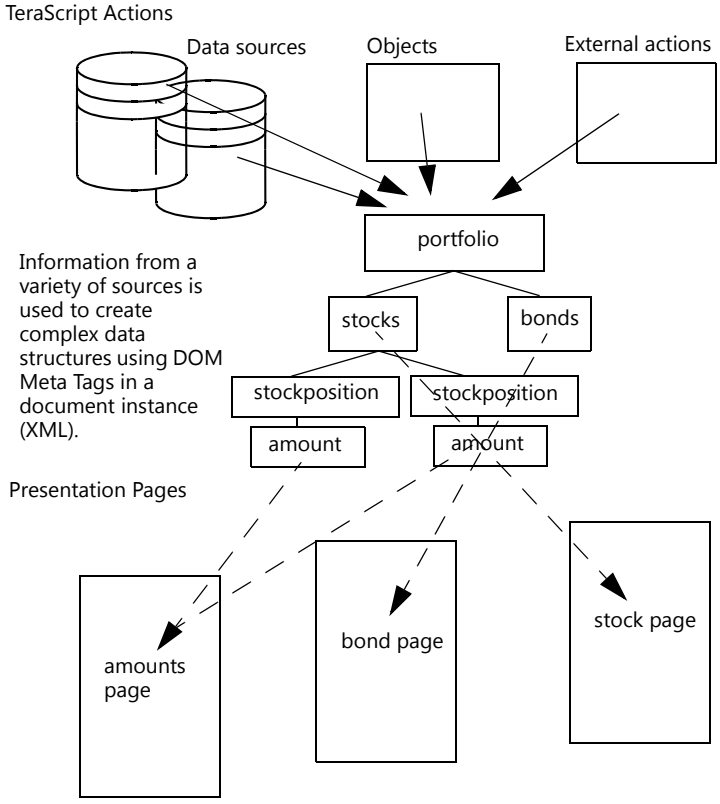
These two different components of a TeraScript application can be used to create complex solutions in TeraScript. Separating business from presentation logic allows changes in the method of processing information, without affecting the method of presenting information, and is especially useful in complex projects. For example, one set of developers could be working on the appearance of the Web pages with proper formatting and design; the other developers could be working on the creation of the data structures to be returned.

TeraScript provides you with a Presentation action that can assist with the separation between presentation and business logic.

While it is not necessary to use XML in the form of TeraScript variables and document instances in order to separate business logic from presentation logic, it can be of enormous assistance because using DOM Meta Tags can easily create complex data structures. The document instance can be seen as an intermediate representation of results from business logic, which can then be translated into HTML for presentation. This is sometimes called the *Presentation Document Object Model*, or PDOM.

The previous section of this chapter on complex data structures shows examples of building up a complex document instance using DOM Meta Tags and retrieving values from that document instance using ELEMENT Meta Tags. The separation of business and presentation logic means that the DOM Meta Tags would be found in the Results HTML of the various actions, in order to build the structure, and the <@ELEMENT...> Meta Tags would be used on the presentation pages to retrieve information from the structure.

The following diagram shows a representation of the data flow between business and presentation logic:



Presentation pages created by the presentation developer and seen by the user. ELEMENT Meta Tags are used to return values from the complex data structure (document instance).

Reading and Writing TeraScript Application Files

For more information, See "XML Format" on page 59.

TeraScript application files are in XML format. This means that you can read these files into a TeraScript variable, creating a document instance. Once the XML has been saved as a TeraScript variable, you can then perform a variety of actions on the document instance: extract information from the document instance, make changes and write out the file, and so on.

For example, the following Meta Tags read in a TeraScript application file, turning the XML into a document instance:

```
<@ASSIGN NAME="myTaf" VALUE=<@DOM VALUE=<@INCLUDE
FILE="Login.taf">>>
```

```
<@ASSIGN NAME="myTaf" VALUE="<@INCLUDE
FILE=Login.taf>">
<@DOMINSERT OBJECT="myDOM">
@@myTAF
</@DOMINSERT>
```

The following Meta Tags return the deployment data sources in the form of an array from the `Login.taf` file (the `DeploymentDSID` attribute lists the name of the deployment data sources for all database-accessing actions):

Using XPointer Syntax:

```
<@ELEMENTATTRIBUTE OBJECT="myTaf"
NAME="DeploymentDSID"
XPOINTER="root().descendants(all)">
```

Using XPath Syntax:

```
<@ELEMENTATTRIBUTE OBJECT="myTaf"
NAME="DeploymentDSID" XPATH="/*">
```

You can create TeraScript application files that generate or modify TeraScript files, using the DTD for TeraScript application files and TeraScript class files. This is useful for advanced TeraScript developers who want to automate the process of generating application files.

Caution Modifying TeraScript application files or TeraScript class files outside of TeraScribe is recommended for advanced users only. Even advanced users should make backups of any files that are to be modified.

Other Uses

XML has many uses. The ability of TeraScript to create and manipulate document instances enables you to do the following:

- Receive, modify, and send Electronic Data Interchange data in XML format.
- Generate XML for presentation on the Internet through the use of Web browsers or plug-ins to Web browsers that can parse and render dialects of XML:
    - generate equations using the Math Markup Language (MathML). For more information on MathML, see:

      `http://www.w3.org/Math/`

    - author multimedia presentations in TeraScript using the Synchronized Multimedia Integration Language (SMIL). For more information on SMIL, see:

http://www.w3.org/AudioVideo/

CHAPTER 6

Configuration Variables

6

Setting TeraScript Options With Configuration Variables

Configuration variables set options controlling the operation of TeraScript Server. This chapter describes the configuration variables and also lists their default values and the scopes in which they are valid.

Configuration variables with scope other than system can be set just like any other variable: use the <@ASSIGN> Meta Tag with the SCOPE attribute set in HTML or in the Assign action when building an application file.

To change system configuration variables, you must first set `configPasswd` with `scope=USER` to match the system configuration variable `configPasswd`, or you can use the `config.taf` application file to set system configuration variables more easily.

TeraScript Server switches (for example, `fileReadSwitch`, `javaSwitch`) are special configuration variables that enable or disable certain TeraScript features.

Configuration variables are saved in the configuration file (`witango.ini`).

A Note on Scope

The description for each configuration variable lists the scopes in which they are valid (for example, "Valid in all scopes" or "System scope only"). If there is an attempt to set a configuration variable in an unregistered scope an error will be generated by the TeraScript Server.

Note Configuration variables are never valid in cookie scope.

For those variables that belong to all or a variety of scopes, adding scope specifications has the following effects:

- `scope=METHOD` sets the configuration variable value for the current method within a TeraScript class file.
- `scope=INSTANCE` sets the configuration variable value for the current instance of a TeraScript class file.
- `scope=REQUEST` sets the configuration variable value for the current application file.
- `scope=USER` sets the configuration variable value to be used with the current user.
- `scope=APPLICATION` sets the configuration variable value to be used in the current TeraScript application.
- `scope=DOMAIN` sets the configuration variable value to be used in the current TeraScript domain.
- `scope=SYSTEM` sets the configuration variable value to be used in TeraScript Server. An administrative password is required to set or change the value of a system configuration variable.

A Note on Default Locations

The following paths are the system defaults under different operating systems for files whose locations are set by TeraScript configuration variables; henceforth, this is called the *configuration directory*.

- **OS X**
  `WITANGO_PATH/Configuration`
- **Windows**
  `WITANGO_PATH\Configuration`
- **Linux**
  `WITANGO_PATH/configuration`

Note WITANGO_PATH refers to the TeraScript Server installation directory. For more information, see "Conventions used in this manual" on page 2.

This affects the following configuration variables:

`appConfigFile`	page 388
`defaultErrorFile`	page 414
`domainConfigFile`	page 417
`headerFile`	page 430
`lockConfig`	page 437
`objectConfigFile`	page 450
`pidFile`	page 455
`timeoutHTML`	page 474
`varCachePath`	page 481

Alphabetical List of Configuration Variables, With Scopes

Configuration Variable	System	Domain	Application	User	Request
absolutePathPrefix	X	X	X	X	X
altUserKey	X	X	X		X
appConfigFile	X				
applicationSwitch	X				
aPrefix	X	X	X	X	X
aSuffix	X	X	X	X	X
cache	X				
cacheIncludeFiles	X				
cacheSize	X				
cDelim	X	X	X	X	X
configPasswd	X		X	X	
cPrefix	X	X	X	X	X
crontabFile	X				
cSuffix	X	X	X	X	X
currencyChar	X	X	X	X	X
customScopeSwitch	X		X		
customTagsPath	X		X		
dataSourceLife	X				
dateFormat	X	X	X	X	X
DBDecimalChar	X	X	X	X	X
debugMode	X	X	X	X	X
decimalChar	X	X	X	X	X
defaultErrorFile	X		X		
defaultScope	X				X
docsSwitch	X		X		
domainConfigFile	X				
domainScopeKey	X				
DSConfig	X				

Configuration Variable	System	Domain	Application	User	Request
DSConfigFile	X				
enableWitangoUserDocs	X				
encodeHTTPResponse	X	X	X	X	X
encodeResultHTML	X	X	X	X	X
externalSwitch	X		X		
fileDeleteSwitch	X		X		
fileReadSwitch	X		X		
fileWriteSwitch	X		X		
headerFile	X		X		
httpHeader	X	X	X	X	X
javaScriptSwitch	X		X		
javaSwitch	X		X		
license	X				
licenseErrorHTML	X				
listenerPort	X				
lockConfig	X				
logArguments	X				
logDir	X				
loggingLevel	X				
logToResults	X	X	X	X	X
mailAdmin	X				
mailDefaultForm	X	X	X	X	X
mailPort	X		X		
mailServer	X		X		
mailSwitch	X		X		
maxActions	X				
maxSessions	X				
noSQLEncoding	X	X	X	X	X
ociLibraryPath	X				

Configuration Variable	System	Domain	Application	User	Request
odbcDmlLibrary	X				
objectConfigFile	X		X		
passThroughSwitch	X		X		
persistantRestart	X				
pidFile	X				
postArgFilter	X	X	X	X	X
queryTimeout	X	X	X	X	X
rDelim	X	X	X	X	X
requestQueueLimit	X				
returnDepth	X				
rPrefix	X	X	X	X	X
rSuffix	X	X	X	X	X
sendUserReferenceKey	X	X	X	X	X
shutdownURL	X				
startStopTimeout	X				
startupURL	X		X		
staticNumericChars	X				
stripCHARs	X	X	X	X	X
TCFSearchPath	X	X	X	X	X
thousandsChar	X	X	X	X	X
threadPoolSize	X				
timeFormat	X	X	X	X	X
timeoutHTML	X				
timestampFormat	X	X	X	X	X
useFullPathForIncludes	X				
userAgent	X	X	X	X	X
userKey	X	X	X		X
validHosts	X				
varCachePath	X				
variableTimeout*	X			X	

Configuration Variable	System	Domain	Application	User	Request
variableTimeoutTrigger*				X	
"webServices"	X				
"webServicesExtns"	X				

*These configuration variables can be used with a custom scope.

absolutePathPrefix

System & Application scope

This configuration variable allows the server administrator to specify a path which limits the File action, External (command line) actions, and attachments to Mail actions. The value of this configuration variable is prepended to the path specified in the File, External, or Mail action.

This configuration variable can be used to set security on file reads, file writes, file deletes, mail attachments, and external actions that invoke the command line. Having the `absolutePathPrefix` set on a system-wide or application-specific basis means that system users or application users cannot access files in directories other than those under a certain directory.

If this configuration variable is left empty the file action path must be a fully qualified path, that it, it must begin either with a drive specification or be in the UNC format (eg. \\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.

altUserKey

See "userKey, altuserKey" on page 478.

appConfigFile

System scope only

Application definitions are read in by TeraScript Server at startup from the path and file determined by this system configuration variable.

The file is by default called `applications.ini` (Windows/UNIX) and resides in the configuration directory. See "A Note on Default Locations" on page 381.

See also

`<@APPKEY>`	page 24
`<@APPNAME>`	page 25
`<@APPPATH>`	page 26
`applicationSwitch`	page 389

applicationSwitch

System scope only

This configuration variable determines whether TeraScript Server supports application scope. When this switch is turned off, TeraScript Server does not support TeraScript applications, and <@APPNAME>, <@APPPATH>, and <@APPKEY> always return empty.

The main reason for this switch is to improve performance when applications are not being used.

Valid values are on and off.

See also

<@APPKEY>	page 24
<@APPNAME>	page 25
<@APPPATH>	page 26
appConfigFile	page 388

aPrefix

Valid in all scopes

This variable sets the prefix character for the entire array when the Meta Tag <@VAR> is used to return the value of an array and convert the array values to text (for example, in Results HTML).

The default value of this variable is <TABLE BORDER="1">, so that returning the values of arrays when arrays are converted to text generates HTML tables.

See also

aSuffix	page 391
cPrefix	page 397
cSuffix	page 401
rPrefix	page 461
rSuffix	page 462

aSuffix

Valid in all scopes

This variable sets the suffix character for the entire array when the Meta Tag <@VAR> is used to return the value of an array and convert the array values to text (for example, in Results HTML).

The default value of this variable is </TABLE>, so that returning the values of arrays when arrays are converted to text generates HTML tables.

See also

aPrefix	page 390
cPrefix	page 397
cSuffix	page 401
rPrefix	page 461
rSuffix	page 462

cache

System scope only

This configuration variable turns the TeraScript Server cache on or off. Possible values are `true` and `false`.

The default value of this variable is `false`.

See also

`cacheSize`	page 394

cacheIncludeFiles

System scope only

This configuration variable turns the TeraScript Server include file caching on and off. The possible values for this variable are `true` and `false`. The default is `true`. This variable only has an effect if the cache configuration variable is set to `true`; see the previous section.

See also

`<@PURGECACHE>` page 224

cacheSize

System scope only

The value of this configuration variable specifies, in bytes, how much of TeraScript memory is used for caching application files and files referenced with `<@INCLUDE>`.

TeraScript Server caches in memory each file it reads, so that subsequent accesses of the same file are faster. The maximum size of the cache is controlled by this configuration variable. When the cache fills up, and TeraScript tries to load an uncached file, any cached included files are purged to make room. If that fails to free enough memory to load the file, the cached application files are purged.

You should set the value of `cacheSize` to a value large enough to accommodate the files that are regularly accessed by TeraScript.

The default value of `cacheSize` is `2000000`.

See also

`cache` page 392

cDelim

Valid in all scopes

This variable sets the default delimiter character between columns for creating arrays with the Meta Tag <@ARRAY>.

The default value of this variable is ",".

See also

rDelim page 458

configPasswd

User, application, and system scopes

This configuration variable sets the password that must be entered for a user to be allowed to change system configuration variables.

When you attempt to set a system configuration variable, TeraScript checks to see if there is a user variable called `configPasswd` that matches the corresponding system variable. If there is, TeraScript lets you change configuration variables. If not, TeraScript returns an error message.

That is, before attempting to set system configuration variables, you must assign the value to the `configPasswd` user variable that matches the system configuration variable `configPasswd`.

When you use the Terascript 6 Administration Application, you are prompted for this password.

`configPasswd` with user scope is used as the password for assignments to configuration variables in application scope. That is, in order to make an assignment to a configuration variable in application scope, you must assign the value to the `configPasswd` user variable that matches the application configuration variable `configPasswd`.

This password is case sensitive.

cPrefix

Valid in all scopes

This variable sets the prefix character for columns (that is, individual data items) of an array. The Meta Tag <@VAR> is used to return the value of an array and convert the array values to text (for example, in Results HTML).

The default value of this variable is <TD>, so that returning the values of arrays when arrays are converted to text generates HTML tables.

See also

aPrefix	page 390
aSuffix	page 391
cSuffix	page 401
rPrefix	page 461
rSuffix	page 462

crontabFile

System scope only

This configuration variable points to the `crontab` file used to set up timed URL processing with TeraScript.

The default value of this configuration variable is empty.

The timed URL execution (*cron*) component of TeraScript Server lets you execute specified URLs at stipulated time intervals. Only HTTP-type URLs are supported. The URLs may point to the local Web and TeraScript Server or a remote one.

TeraScript Server discards the Result HTML of an executed URL.

Both the specification file and the execution mechanism are modeled after the Unix cron(1) daemon. The specifications are stored in a separate file (conventionally called crontab), which TeraScript Server reads when it starts up. If this file is modified, you must restart TeraScript Server or use <@RELOADCONFIG> to load the new settings.

The timed query module of TeraScript Server becomes active once per minute, and executes, one by one, all the URLs that satisfy the specification conditions between the previous and present activation.

The full path to the crontab file must be specified in the system configuration variable crontabFile. The following section discusses the format of the crontab file.

Format of the crontab File

A crontab file consists of lines of six fields each. The fields are separated by spaces or tabs. The following table is a summary of these fields (in this order):

Field number	Field name	Valid values
1	minute	0-59
2	hour	0-23
3	day of the month	1-31
4	month of the year	1-12
5	day of the week	0-6 (0=Sunday)
6	URL	HTTP-type URL

Time Fields (1-5)

The first five fields are numeric patterns. Each of these patterns may be either of the following:

- An asterisk (meaning all valid values)
- A list of elements separated by commas. An element is either a number or two numbers separated by a dash (meaning an inclusive range).
- The following two example values are identical:

  2-4,7,9-12

  2,3,4,7,9,10,11,12

Note The specification of days may be made by two fields (day of the month and day of the week) or both.

URL Field (6)

The sixth field is a URL executed by TeraScript Server at the specified time(s). It must be in HTTP-type URL form, which is:

```
http://host:port/path?search-arguments
```

- `host` may be specified as fully qualified hostname or Internet address, and may not be omitted.
- `port` may be omitted, in which case the default HTTP port (80) is assumed.
- `path` and `search-arguments` may be omitted, in which case a request for the main page is sent.

Example of the crontab File

The following is an example of a crontab file. Lines starting with a hash or pound sign (#) are treated as comments; TeraScript Server ignores these lines.

```
#
# Example crontab file
#
# Run this application file all the time (every minute)
* * * * * http://127.0.0.1/query/null.taf
# Gather statistics every hour, Monday through Friday,
# 10am to 6pm
1 10-18 * * 1-5 http://127.0.0.1/query/stats.taf
# Run this query every minute, Monday through Friday,
# 10am to 6pm
* 10-18 * * 1-5 http://127.0.0.1/query/stats.taf
```

```
# Clean up database locks every Sunday =and= on 1st and
# 15th of the month
0 1 1,15 * 0 http://127.0.0.1/query/dbclean.taf
# Run this application file at noon on 14th of July
0 12 14 7 * http://127.0.0.1/query/14-Juillet.taf
```

Note The last line of the crontab file must end with a carriage return.

cSuffix

Valid in all scopes

This variable sets the suffix character for columns in an array that is returned when the Meta Tag <@VAR> is used to return the value of an array and convert the array values to text (for example, in Results HTML).

The default value of this variable is </TD>, so that returning the values of arrays when arrays are converted to text generates HTML tables.

See also

aPrefix	page 390
aSuffix	page 391
cPrefix	page 397
rPrefix	page 461
rSuffix	page 462

currencyChar

Valid in all scopes

(request scope invalid when staticNumericChars=true)

The value of this configuration variable tells TeraScript Server what character string is used as the currency symbol in money values (for example, in the USA and Canada, the dollar sign ($) is used). Values up to three characters in length may be assigned to `currencyChar`. If a longer value is assigned, only the first three characters are used.

TeraScript Server uses this value in order to properly evaluate numbers in conditional comparisons (for example, Branch action, `<@IF>`, `<@IFEQUAL>` and `<@ISNUM>` Meta Tags) and in calculations performed with `<@CALC>`; that is, it recognizes that strings that start or end with these characters are to be treated as numeric and not text.

The setting is also used when TeraScript Server is constructing SQL for Search, Insert, Update, and Delete actions. TeraScript automatically removes the character string specified by `currencyChar` from any values specified for numeric columns. Use the `<@DSNUM>` Meta Tag to perform the same function on numbers you specify in Direct DBMS actions.

The default value of `currencyChar` is `$`.

On Macintosh, the default is the corresponding setting in the *Numbers* control panel on the server computer. You may always revert to the default setting by assigning an empty value to this configuration variable.

currencyChar and Scope

For more information, see "staticNumericChars" on page 467.

When `staticNumericChars` has the value `true` (the default), changing the value of `currencyChar` has no effect during the execution of an application file. Changes to `currencyChar` in user, domain, or system scope take effect with the *next* application file execution; as a consequence, changes to `currencyChar` in request scope have no effect.

When `staticNumericChars` has the value `false`, `currencyChar` works with scope in the standard way.

See also

`DBDecimalChar`	page 410
`decimalChar`	page 412
`<@DSDATE>`	page 119
`<@DSNUM>`	page 121
`<@DSTIME>`	page 119

<@DSTIMESTAMP> page 119
staticNumericChars page 467
thousandsChar page 470

customScopeSwitch

System & Application scope

This configuration variable determines whether custom scopes are allowed. The possible values for this variable are on and `off`.

When set to false, an error is generated if an application file uses a variable scope that is not one of the built-in variable scopes in TeraScript (that is, method, instance, request, user, cookie, domain, application, or system scope).

A system scope value of `off` for this configuration variable overrides an application scope value of on.

customTagsPath

System & Application scope

Custom tag definitions are read in by TeraScript Server at startup from a directory determined by this configuration variable. This configuration variable can have different values for application and system scope; that is, users can create different sets of custom tags for each application, or custom tags that apply to all of TeraScript Server.

For more information on custom tags, see "Using Custom Meta Tags" on page 306.

The default value of this variable points to the `CustomTags` directory under the configuration directory. See "A Note on Default Locations" on page 381. (For example, this is by default: `TERASCRIPT_PATH\CustomTags\` under Windows, and, TERASCRIPT-PATH/customtags on Unix).

Any files in this directory and any subdirectories that contain custom tag definitions are read in and used by TeraScript Server.

See also

`<@CUSTOMTAGS>`	page 88
`<@RELOADCUSTOMTAGS>`	page 231

dataSourceLife

System scope only

The value of this configuration variable indicates how long TeraScript Server keeps open an unused connection to a data source. This variable is specified in minutes. When the time out period is exceeded, the connection to the data source is closed. Each time a data source connection is used, its timeout timer is reset to zero.

When this configuration variable is set to zero, data source connections are always closed immediately after use. Multiple actions in the same execution using the same data source use the same connection; that is, the connection is not closed until the end of the application file execution.

The default value is `30` (minutes).

dateFormat, timeFormat, timestampFormat

Valid in all scopes

These configuration variables allow you to specify the formats for displaying and entering date, time, and timestamp values. The formats determine the default display formats of retrieved database values as well as those returned by the <@CURRENTDATE>, <@CURRENTTIME>, and <@CURRENTTIMESTAMP> Meta Tags. Date, time, and timestamp values specified in Update and Insert actions, and those in criteria values must match the formats specified in these configuration variables. TeraScript converts these values to the formats required by the database.

Note On Macintosh, the default values for `dateFormat`, `timeFormat`, and `timestampFormat`, if they are not explicitly set, come from the *Date & Time* control panel of the computer running TeraScript.

For more information, see "DATETIME" on page 14.

Date and Time Formatting Codes

Code	Description
%a	abbreviated weekday name
%A	full weekday name
%b	abbreviated month name
%B	full month name
%c	local date and time representation
%d	day of month (01–31)
%H	hour (24 hour clock)
%I	hour (12 hour clock)
%j	day of the year (001–366)
%m	month (01–12)
%M	minute (00–59)
%p	local equivalent of AM or PM
%S	second (00–59)
%U	week number of the year (Sunday= first day of week) (00–53)
%w	weekday (0–6, Sunday is zero)
%W	week number of the year (Monday = first day of week) (00–53)
%x	local date representation
%X	local time representation
%y	year without century (00–99)
%Y	year with century
%%	% sign

Examples

If the following date and time formats were used on the twenty-eighth of July, 1998, at 6:30 PM:

```
%A, %B %d, %Y       returns "Sunday, July 28, 1998"
%m/%d/%Y            returns "07/28/1998"
%H:%M:%S            returns "18:30:00"
%I:%M %p            returns "6:30 PM"
```

Note If a date format string contains %Y, but the value for the year is two-digit, the following centuries are assumed, if appropriate:

Value	Century
00-36	2000s
37-99	1900s

That is, a two-digit year of 99 is evaluated as 1999, and a two-digit year of 00 is evaluated as 2000.

The default values of the configuration variables are given in the following table:

Configuration Variable	Default Value
dateFormat	%m/%d/%Y
timeFormat	%H:%M:%S
timestampFormat	%m/%d/%Y %H:%M:%S.

DBDecimalChar

Valid in all scopes

The value of this configuration variable tells TeraScript Server what decimal character ODBC data sources require in numbers. This value may be determined by the ODBC driver, the database vendor's client software, or the DBMS server. You must make sure you set this configuration variable appropriately for the ODBC data sources you are accessing with TeraScript Server.

If you use ODBC data sources requiring different decimal characters, you may use the `DBDecimalChar` with `scope=REQUEST` to change this setting temporarily while accessing a specific data source.

The setting of `DBDecimalChar` is used when TeraScript Server is constructing SQL for Search, Insert, Update, and Delete actions that use ODBC data sources. If necessary—for example, when `DBDecimalChar` differs from `decimalChar`—TeraScript automatically converts values specified for numeric columns to use the decimal character specified in `DBDecimalChar`. Use the `<@DSNUM>` Meta Tag to perform the same function on numbers you specify in Direct DBMS actions using an ODBC data source.

The default value of `DBDecimalChar` is a period (".").

See also

`currencyChar`	page 402
`decimalChar`	page 412
`thousandsChar`	page 470
`<@DSDATE>`	page 119
`<@DSTIME>`	page 119
`<@DSTIMESTAMP>`	page 119
`<@DSNUM>`	page 121

debugMode

Valid in all scopes

This configuration variable sets whether debug mode is on.

The default value of this variable is `appFileSetting`, which allows the application file setting of debug mode (the checkbox) to set whether a particular application file has debug mode on or off.

Other possible values are `forceOn` and `forceOff`, which override any application file settings (that is, overrides the debug checkbox in the application file).

This configuration variable does NOT affect the output to the log file. It is used exclusively to manage the debug output in the result HTML. The variable is evaluated once for every action executed by the TeraScript Server.

decimalChar

Valid in all scopes

(request scope invalid when staticNumericChars=true)

The value of this configuration variable tells TeraScript Server what character is used as the decimal character in numbers. In the US, the period, ".", is normally used for this purpose. Only a single character may be assigned to `decimalChar`. If a longer value is assigned to `decimalChar`, only the first character is used.

TeraScript Server uses this value in order to properly evaluate numbers in conditional comparisons (Branch action, `<@IF>`, and `<@IFEQUAL>`) and in calculations performed with `<@CALC>`. The setting is also used when TeraScript Server is constructing SQL for Search, Insert, Update, and Delete actions. If necessary, TeraScript automatically converts values specified for numeric columns to use the decimal character required by the DBMS.

Use the `<@DSNUM>` Meta Tag to perform the same function on numbers you specify in Direct DBMS actions.

TeraScript uses this setting to format any numeric values retrieved from a data source.

The default value of `decimalChar` is "." (a period).

On Macintosh, the default is the corresponding setting in the *Numbers* control panel on the server computer. You may always revert to the default setting by assigning an empty value to this configuration variable.

decimalChar and Scope

For more information, see "staticNumericChars" on page 467.

When `staticNumericChars` has the value `true` (the default), changing the value of `decimalChar` has no effect during the execution of an application file. Changes to `decimalChar` in user, domain, or system scope take effect with the *next* application file execution; as a consequence, changes to `decimalChar` in request scope have no effect.

When `staticNumericChars` has the value `false`, `decimalChar` works with scope in the standard way.

See also

`currencyChar`	page 402
`DBDecimalChar`	page 410
`<@DSDATE>`	page 119
`<@DSNUM>`	page 121
`<@DSTIME>`	page 181

`<@DSTIMESTAMP>` page 119
`staticNumericChars` page 467
`thousandsChar` page 470

defaultErrorFile

System & application scope

This configuration variable specifies a path to a file on the TeraScript Server machine. TeraScript uses the contents of this file as the error message returned to a user whenever an error condition occurs within an application file (unless you have specified Error HTML within the application file itself).

The default error file is `error.htx`. This file is in HTML format and may contain Meta Tags. You can edit the file with a text or HTML editor.

You can set different default error files for TeraScript applications by assigning to this variable in application scope.

The default value of this configuration variable are the *ErrorMessage* and *HelpMessage* parts of the <@ERROR> Meta Tag, when those values are not empty.

Note The *HelpMessage* text is NOT written into the log file.

defaultScope

Request or system scope

This configuration variable sets the default scope that variables are created with when the Assign action or the <@ASSIGN> Meta Tag are used without specifying a scope.

The default value of `defaultScope` is `REQUEST`.

docsSwitch

System & Application scope

This configuration variable determines whether TeraScript Server allows the use of the <@DOCS> Meta Tag, which returns the contents of an application file. Valid values are on (the default) and off. A switch is provided because examining the contents of any application file could potentially be a security issue.

A system scope value of off for this configuration variable overrides an application scope value of on.

See also

<@DOCS> page 106

domainConfigFile

System scope only

This configuration variable points to a file where TeraScript domains are set up. These TeraScript domains are used as the key value for domain scope in TeraScript.

The default value of this configuration variable is a file called `domains.ini` (Windows/UNIX) in the configuration directory. See "A Note on Default Locations" on page 381.

See also

`domainScopeKey`	page 418
`<@DOMAIN>`	page 108

domainScopeKey

System scope only

Meta Tags evaluated

This configuration variable sets the key for the domain scope; that is, what value TeraScript uses in order to determine which TeraScript domain a request originated from and the value it uses as a key to find domain variables internally. The value for this configuration variable may contain Meta Tags. The tags are substituted each time the variable is used by TeraScript Server.

TeraScript uses any TeraScript domains as the domain scope key, if any are set up; if none are set up, it defaults to the domain name (base URL or IP address).

For more information, see "<@CGIPARAM>" on page 57, "<@CIPHER>" on page 64, and "<@LOGMESSAGE>" on page 197.

The default value is `<@CIPHER ACTION=HASH STR="<@LOWER <@DOMAIN>>">`. This uses a lower-cased, encrypted form of the TeraScript domain (if any are set up), or defaults to the domain name (base URL or IP address) retrieved with `<@CGIPARAM SERVER_NAME>`.

The value of the `domainScopeKey` cannot be greater than 32 characters. `<@CIPHER action=hash>` always results in a 32 character string.

When you assign a value to `domainScopeKey`, you must tell TeraScript Server to evaluate the Meta Tag only when domain variables need to be keyed. This is done with the `<@LITERAL>` Meta Tag.

For more information, see "<@LITERAL>" on page 195.

For example, the syntax of the assignment to `domainScopeKey` of its default value would be as follows:

```
<@ASSIGN NAME=domainScopeKey VALUE=
<@LITERAL VALUE="<@CIPHER ACTION=HASH STR='<@LOWER
<@DOMAIN>>'>">>
```

See also

`<@DOMAIN>` page 108

DSConfig

System scope only

This configuration variable allows you to modify some TeraScript data source parameters which may be required to tune database performance on an individual basis. These parameters include whether the data source driver is thread-safe, and the maximum number of connections allowed to the data source.

`DSConfig` contains an array, and is not specified within the TeraScript Server configuration file. The contents of the `DSConfig` array are written to and read from a file.

For more information on the format and location of this file, see "DSConfigFile" on page 421.

By default, this file is called `dsConfig.ini` (Windows and UNIX). You can also set the name and location of the file to something other than the default by modifying the value of the `DSConfigFile` configuration variable.

Caution Do not edit the `dsConfig.ini` file directly when TeraScript Server is running. Either stop TeraScript Server and edit this file, or use the Terascript 6 Server Administration application or your own application files to create or modify the `DSConfig` array, which is then automatically written out to the `dsConfig.ini`file.

It is recommended that you use the `config.taf` application file to modify the values of this variable.

When `DSConfig` is updated, changes are written immediately to the file specified by `DSConfigFile`.

For more information on the structure of the data source configuration file, see "DSConfigFile" on page 421.

The `DSConfig` array has the following structure:

Row 0	*type.name*	*type.name*
Row 1 (*maxconnections*)	n	n

The `type` parameter defines the type of data source: ODBC, Oracle or JDBC. The `name` parameter defines the name of the data source, the Oracle alias or connect string.

The `maxconnections` parameter defines the maximum number of connections that TeraScript Server makes to the data source. The default is '0' (no limit).

Setting `maxconnections` to a value other than zero can be useful if you have a limited user license for your database server. For example, if you have a five-user license only, TeraScript Server may use all of the connections when running application files. Setting the `maxconnections` value to less than five allows other users to connect to the database while TeraScript Server is also running.

However, you should not set `maxconnections` to a value which is too low for your data source setup; for example, if `maxconnections` is set to "2" and TeraScript Server has two open database connections, the next user that tries to connect via TeraScript Server to a database may experience a wait until the connection is free, or the query may time out when the `queryTimeout` value is reached.

DSConfigFile

System scope only

This configuration variable contains the name of the data source configuration file to read from and write to. The default path is to `dsConfig.ini` in the configuration directory. See "A Note on Default Locations" on page 381.

A global connection pool feature has been added to TeraScript Server 6 to prevent TeraScript from consuming all available connections from a data source with connection limitations. The DSConfig.ini file had previously supported a MAXCONNECTIONS parameter to restrict TeraScript from making too many connections per data source. Now with the global pool feature you can restrict connections among multiple data sources.

Simply add the parameter GLOBALPOOL=# to each data source stanza that you wish to have included to the global pool. The # should be the maximum number of connections allowed to the global pool. There is only one global pool available in each server instance. Additionally, you can continue to use MAXCONNECTIONS to tune the connections per each data source.

The data source configuration file is structured as follows:

```
[Data Sources]
myFirstDS=comment on first ds
mySecondDS=comment on second ds

[myFirstDS]
TYPE=ODBC
MAXCONNECTIONS=5
GLOBALPOOL=8

[mySecondDS]
TYPE=ODBC
MAXCONNECTIONS=5
GLOBALPOOL=8
```

In this configuration, both data sources can open up to 5 connections to the database server, however, their cumulative connection count will not exceed 8. The value given in the GLOBALPOOL parameter should be the same for all data sources participating the in the global pool.

Using the global pool, you can access multiple data sources which require or operate better with single-threaded (sequential) requests. Oterro, for example, requires a GLOBALPOOL=1 for proper operation.

For more information on the parameters that are set in this file, see "DSConfig" on page 419.

Stanza names must be unique in this file. Stanza names are one of the following: the name of the ODBC data source, the Oracle alias or connect string, or JDBC data source.

When TeraScript Server starts up, if `DSConfigFile` contains a valid path to an existing file, the contents of the file are used to set up the `DSConfig` variable.

encodeHTTPResponse

Valid in all scopes

This configuration variable 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 reponse to the browser will have high ascii characters encoded.

The default value of this variable is `true`.

See also

`encodeResults` page 424

`encodeResultsHTML` page 425

encodeResults

Valid in all scopes

This configuration variable was discontinued with the release of Witango Server v5.5.

See also

encodeHTTPResponse page 423

encodeResultsHTML

Valid in all scopes

`ENCODERESULSTHTML` allows the programmer or system administrator control of the html encoding that occurs as results are pushed onto the results buffer during TAF execution. This encoding converts the following characters:

<	&lt;
>	&gt;
"	&quot;
&	&amp;

This variable can be set to `true` or `false`. The default value is `false`.

See also

`encodeHTTPResponse` page 423

externalSwitch

System & Application scope

This configuration variable determines whether TeraScript Server allows External actions.

A system scope value of `off` for this configuration variable overrides an application scope value of on.

Valid values are on and `off`.

fileDeleteSwitch

System & Application scope

This configuration variable determines whether TeraScript Server allows the deletion of external files using the File action.

A system scope value of `off` for this configuration variable overrides an application scope value of on.

Valid values are on and `off`.

See also

`absolutePathPrefix` page 386
`fileReadSwitch` page 428
`fileWriteSwitch` page 429

fileReadSwitch

System & Application scope

This configuration variable determines whether TeraScript Server allows the reading in of external files using the File action or the <@INCLUDE> Meta Tag.

A system scope value of `off` for this configuration variable overrides an application scope value of on.

Valid values are on and `off`.

See also

`absolutePathPrefix`	page 386
`fileDeleteSwitch`	page 427
`fileWriteSwitch`	page 429
`<@INCLUDE>`	page 168

fileWriteSwitch

System & Application scope

This configuration variable determines whether TeraScript Server allows writing out to external files using the File action.

A system scope value of `off` for this configuration variable overrides an application scope value of on.

Valid values are on and `off`.

See also

`absolutePathPrefix` page 386
`fileDeleteSwitch` page 427
`fileReadSwitch` page 428

headerFile

System & Application Scope

This configuration variable sets the file to be used as the HTTP header that is returned every time a reply is sent to a Web browser. The default value of this configuration variable is `header.htx`.

You can set different header files for TeraScript applications by assigning to this variable in application scope.

httpHeader

Valid in all scopes

This configuration variable determines the HTTP header used when TeraScript Server returns results to a user. The HTTP header sends information to a Web browser about the request: whether it was successful and what kind of information is being returned. It can also be used to redirect the Web browser to a different URL.

Note Changes made to this configuration variable are not saved; it reverts to the value specified in the file pointed to by the `headerFile` each time you start TeraScript Server (the default). To make a permanent change to the HTTP header, use `headerFile`.

For more information, see “headerFile” on page 430.

The value of `httpHeader scope=request` determines the content of the HTTP header for the result of the current application file execution. You may set this at any point in the file execution.

javaScriptSwitch

System & Application scope

This configuration variable determines whether TeraScript Server allows the execution of JavaScript in TeraScript Server (that is, JavaScript delineated with the <@SCRIPT> tag or using the Script action).

Note This is different from JavaScript that is passed onto and executed by the Web browser using the <SCRIPT> HTML tag, which is not affected by this configuration variable.

A system scope value of `off` for this configuration variable overrides an application scope value of on.

Valid values are on and `off`.

javaSwitch

System & Application scope

This configuration variable determines whether TeraScript Server allows the execution of Java.

A system scope value of `off` for this configuration variable overrides an application scope value of on.

Valid values are on and `off`.

license

System scope only

This configuration variable contains your TeraScript Server license key. The server runs only if a valid license key is entered.

licenseErrorHTML

System scope only

This configuration variable defines the path to a file containing HTML to return when the maximum number of sessions allowed by the TeraScript Server license is exceeded. By default, this variable points to the `licError.htx` file in the `Configuration` folder under the folder where TeraScript is installed. If you do not specify a file, TeraScript returns a built-in default error message.

listenerPort

System scope only


Windows and Linux only.

This configuration variable sets the port number used by TeraScript Server to listen for requests from the TeraScript CGI. This number can be any valid port number that is not currently in use on your system. (Various UNIX operating systems and applications reserve ports.)

The default value is `18160`.

Note By default the `LISTENERPORT` variable is set to 18160 to prevent conflicts with previous versions of TeraScript Server which use different ports. This means that previous versions of TeraScript servers can be running at the same time on a single server during the upgrade phase of your service.

lockConfig

System scope only

If this configuration variable is present and set to 'true', the TeraScript Server configuration file (`witango.ini`) is set to read-only; that is, changes made to configuration variables within the course of an application file execution are not written out to disk. They must be made manually to the configuration file.

If this configuration variable is not present, or is present and set to `false`, there is no effect.

This configuration variable cannot be set with the `config.taf` application file: you must manually add the following entry to the TeraScript Server configuration file:

```
LOCKCONFIG=true
```

logArguments

System scope only

This Configuration Variable exists to enable a programmer to stop the logging of sensitive information to the `witango.log`. The setting of this variable will determine whether searcharguments and postarguments are written to log files.

The setting of this configuration variable will override all logging levels.

`logArguments` can be set to `TRUE` or `FALSE`.

The default value for `logArguments` is `false`.

See also

`loggingLevel` page 440
`logToResults` page 441
`loggingLevel` page 440

logDir

System scope only

This configuration variable sets the directory used for logging. The log directory should be unique for every TeraScript Server running on the same machine.

The default value of this variable is `{default path}log.{name of server}`; for example:

On Windows:

TERASCRIPT_PATH\Configuration\log.Terascript_Server_6

On Linux:

TERASCRIPT_PATH/configuration\log.Terascript_Server_6

loggingLevel

System scope only

This configuration variable controls what information is written to the `Witango.log` file. There are five values that can be assigned to this configuration variable, corresponding the five possible levels of logging.

The following table lists each value and describes what information is logged.

Level	Information Logged
0	None
1	application file execution, search and post argument values.
2	LogLevel1 information plus application file actions.
3	LogLevel2 information plus generated SQL, variable and action result values.
4	LogLevel3 information plus Results HTML.

Higher logging levels may affect the performance of your Web server, particularly if there is a lot of traffic. You may want to use high logging levels (particularly `3` and `4`) only while you need to track down problems with your application files.

The default value of `loggingLevel` is `NoLogging`.

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$loggingLevel* system variable is assigned an incorrect value in an application file, the request will fail with an error.

See also

`debugMode` page 411
`logToResults` page 441
`logArguments` page 438

logToResults

Valid in all scopes

Controls whether the logging information execution is returned with Results HTML. This option is useful for debugging. When set to `true`, all the information written to the TeraScript log file for an application file execution is also returned with the Results HTML. The default value of `logToResults` is `false`.

The current setting of `loggingLevel` configuration variable determines the amount of information logged. To see logging information for a particular application file execution, assign `true` to this configuration variable with `scope=REQUEST`.

Note Selecting the Debug mode option in the application file window is equivalent to setting `logToResults scope=REQUEST` to true and `loggingLevelscope=REQUEST` to `LogLevel3`. Because these variables are set with request scope, they are only set to these values for the duration of the file execution.

See also

`debugMode`	page 411
`logArguments`	page 438
`loggingLevel`	page 440

mailAdmin

System scope only

This configuration variable specifies the e-mail address of the administrator to whom the messages are sent when the maximum number of sessions is exceeded.

mailDefaultFrom

Valid in all scopes

This configuration variable determines the default `From` value for e-mail messages sent using the Mail action of TeraScript.

This default is overridden by any value you type in the **From** field of the Mail action.

This configuration variable is also used as the default value of the `FROM` attribute of the `<@URL>` tag, and the `From` value in HTTP requests generated by TeraScript's timed URL processing, startup/shutdown URLs, and the URL specified in `variableTimeoutTrigger`.

See also

`mailPort`	page 444
`mailServer`	page 445

mailPort

System & Application Scope

This configuration variable specifies the port that the e-mail server specified by `mailServer` uses.

The default value of this variable is `25`.

See also

`mailDefaultFrom` page 443
`mailServer` page 445

mailServer

System & Application Scope

This configuration variable sets the SMTP e-mail server that is used for messages sent with the Mail action.

See also

`mailDefaultFrom` page 443
`mailPort` page 444

mailSwitch

System & Application scope

This configuration variable determines whether TeraScript Server allows e-mail messages to be generated within TeraScript using the Mail action.

A system scope value of `off` for this configuration variable overrides an application scope value of on.

Valid values are on and `off`.

See also

`mailDefaultFrom`	page 443
`mailPort`	page 444
`mailServer`	page 445

maxActions

System scope only

If this configuration variable is set to a positive number (the default is zero), the number of TeraScript actions executed so far by a query is checked against the value of this variable. If the number of actions exceeds the value, the query aborts and returns an error.

Note A looping query always aborts when the execution time exceeds the time specified in the configuration variable `queryTimeout`. `maxActions` provides finer control over infinite loops.

See also

`queryTimeout` page 457

maxSessions

System scope only

Macintosh only.

This system configuration variable determines the maximum number of sessions TeraScript Server opens for a particular data source host. It accepts any positive integer as a value. A value of zero indicates no maximum.

Note This system configuration variable is applicable only to DAM data sources under Macintosh. It has no effect on connections made to other data source types or on other operating systems.

The default value of `maxSessions` is zero, indicating that there is no limit on the number of data source connections that TeraScript Server makes to a particular DAM host.

noSQLEncoding

Valid in all scopes

This configuration variable determines whether text in Direct DBMS actions is SQL-encoded by default (single quote characters doubled). The default value is `false`. Setting the value to true turns off automatic SQL-encoding in Direct DBMS actions.

If `noSQLEncoding` is set to `true`, you can use the `ENCODING=SQL` attribute on most value-returning Meta Tags to SQL-encode the value returned by that Meta Tag.

See also

Encoding Attribute page 8

objectConfigFile

System & Application scope

Object configuration information—specifically, which objects are allowed to be run on TeraScript Server—is read from the file pointed to by this configuration variable. The default value of this variable is a file called `objects.ini` (Windows/UNIX) in the configuration directory. See "A Note on Default Locations" on page 381.

ociLibPath

System & Application scope

Linux Only.

This configuration variable is used by the TeraScript Server to set a search path for the Oracle Call Interface shared library. When this configuration variable is set, the TeraScript Server will use the path described by the variable to search for the shared library, ahead of any other locations.

odbcDmlLibrary

System Scope

Linux Only.

The system variable is used to specify the path to the ODBC Driver Manager Shared library.

The default value is none.

passThroughSwitch

System & Application scope

The use of Meta Tags in data source specifications is by default enabled in TeraScript. The `passThroughSwitch` option controls whether Meta Tags (such as `<@POSTARG>`) are permitted in all fields when connecting to a data source using an application file. If this switch is disabled, all data source parameters—type, name, database, user name, and password—must be hard coded.

A system scope value of `off` for this configuration variable overrides an application scope value of on.

Valid values are on and `off`.

persistentRestart

System scope only


Windows and Linux only.

This configuration variable controls how the server handles an automatic restart. An automatic restart is initiated when TeraScript detects a problem with servicing requests.

When set to `true`, the server first attempts to completely shut down the running server before restarting a new one. All variables in use at the time of the shutdown are preserved.

When set to `false`, a new server is started immediately, even before the old one is stopped. This setting ensures high server availability, but variables from the old server instance are not available in the new one. The default value is `true`.

pidFile

System scope only


Windows and Linux only.

This configuration variable sets the location of a file is used to track the TeraScript Server process. It should have a unique name for every TeraScript Server running on the same machine. The default value is `{default path}pid.{name of server}`.

postArgFilter

Valid in all scopes

The `postArgFilter` configuration variable accepts a value containing one or more characters, each of which is automatically removed from post argument values received by TeraScript Server. The characters can be specified by their ASCII number using the `<@CHAR>` tag.

For more information, see "<@CGI>" on page 56.

This configuration variable is useful for automatically removing the linefeeds that some Web browsers use for ending lines entered into `<TEXTAREA>` form fields, and that appear as boxes in Macintosh database applications. To use `postArgFilter` for this purpose, assign `<@CHAR 10>` to it.

The default value of `postArgFilter` is empty.

queryTimeout

System scope only

This configuration file variable causes queries that exceed the specified number of seconds to time out and return the HTML page specified in `timeoutHTML`. This variable is specified in seconds.

The default value of `queryTimeout` is `300`.

See also

`timeoutHTML` page 474

rDelim

Valid in all scopes

This variable sets the default delimiter character between rows for creating arrays with the Meta Tag <@ARRAY>.

The default value of this variable is "`;`".

This variable is valid in all scopes.

See also

`cDelim` page 395

requestQueueLimit

System scope only


Windows and Linux only.

This variable allows customization of the size of the queue used to hold incoming requests. By default, the value of this configuration variable is `0`, which indicates no maximum. If your TeraScript Server typically processes lengthy requests, then setting a value for the queue size can prevent TeraScript Server from getting to a point where the queue contains so many items that it cannot process them before the user's Web browser times out.

returnDepth

System scope only

This configuration file option sets the maximum number of branch "levels" that you can have in a TeraScript application file. This applies to Branch actions that have the Return option set. It specifies the number of returns that can be outstanding at any time. If this limit is exceeded during an application file execution, an error occurs.

This configuration variable also specifies the number of call method "levels" when method calls are made on TeraScript class files, which in turn call other TeraScript class files.

The default value is `20`. Setting this configuration variable to a larger value may increase the memory requirements of TeraScript Server.

rPrefix

Valid in all scopes

This variable sets the prefix character for array rows that is returned when the Meta Tag <@VAR> is used to return the value of an array and convert the array values to text (for example, in Results HTML).

The default value of this variable is <TR>, so that returning the values of arrays when arrays are converted to text generates HTML tables.

See also

`aPrefix`	page 390
`aSuffix`	page 391
`cPrefix`	page 397
`cSuffix`	page 401
`rSuffix`	page 462

rSuffix

Valid in all scopes

This variable sets the suffix character for array rows that is returned when the Meta Tag <@VAR> is used to return the value of an array and convert the array values to text (for example, in Results HTML).

The default value of this variable is </TR>, so that returning the values of arrays when arrays are converted to text generates HTML tables.

See also

aPrefix	page 390
aSuffix	page 391
cPrefix	page 397
cSuffix	page 401
rPrefix	page 461

sendUserReferenceCookie

Valid in all scopes

`SENDUSERREFERENCECOOKIE` controls the sending of the TeraScript user reference cookie to control session management. The variable 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

shutdownUrl

System scope only

This configuration variable contains the HTTP URL to be requested when TeraScript Server shuts down.

The default value is `empty`.

See also

`startStopTimeout`	page 465
`startupUrl`	page 466

startStopTimeout

System scope only

This configuration variable determines how long TeraScript Server waits for a response from the URLs that are called when TeraScript Server shuts down or starts up. The value is specified in seconds.

The default value is `60`.

See also

`shutdownUrl`	page 464
`startupUrl`	page 466

startupUrl

System & Application Scope

This configuration variable contains the HTTP URL, if any, that is requested when TeraScript Server or an application starts up.

The default value is `empty`.

You can set different startup URLs for TeraScript applications by assigning to this variable in application scope.

See also

`shutdownUrl` page 464
`startStopTimeout` page 465

staticNumericChars

System scope only

This configuration variable determines when TeraScript Server checks for changes to the configuration variables that determine the thousands, decimal, and currency characters used for numerical evaluation.

The default value of `true` means TeraScript Server obtains these characters from the `thousandsChar`, `decimalChar`, and `currencyChar` configuration variables at the *beginning* of each application file execution *only*. Any changes to the user, domain, and system scope variables take effect with the *next* application file execution. Request scope for these configuration variables is never used when `staticNumericChars` has the value `true`.

When `staticNumericChars` has the value `false`, any changes to the `thousandsChar`, `decimalChar`, and `currencyChar` configuration variables in any scope take effect immediately.

Tip There is a significant performance benefit to a setting of `true` for this configuration variable. Use a setting of `false` only if you must support different numeric formats over the course of a single application file execution.

stripCHARs

Valid in all scopes

This configuration variable sets whether CHAR (fixed-length text field) and VARCHAR data from data sources is automatically stripped of trailing spaces. Possible values are `true` and `false`.

The default value is `true`.

TCFSearchPath

Valid in all scopes

Meta Tags evaluated

This configuration variable is used to define the search path for TeraScript class files on TeraScript Server. The value for this configuration variable may contain Meta Tags. The tags are substituted each time the variable is used by TeraScript Server. This configuration variable contains a semi-colon separated list of Web server document root relative paths in which to look for TeraScript class files. For example, `TCFSearchPath` may contain the following:

```
MyApp/TCFs/Logon/;MyApp/TCFs/GuestBook/;FoneList/
Objects/;DougApp/OtherStuff/MyObjects/
```

The default value of `TCFSearchPath` is `<@CLASSFILEPATH>; <@APPFILEPATH>`, which means that TeraScript class files are searched for in the directories that are returned by these Meta Tags.

Subfolders of the specified folders are not searched; each subfolder must be specified separately in order to have TeraScript find TeraScript class files there.

See also

`<@APPFILEPATH>` page 23
`<@CLASSFILEPATH>` page 70

thousandsChar

Valid in all scopes

(request scope invalid when staticNumericChars=true)

The value of this configuration variable tells TeraScript Server what character is used as the thousands separator in numbers. For example, in the US, the comma (",") is normally used for this purpose. Only a single character may be assigned to `thousandsChar`. If a longer value is assigned to it, only the first character is used. The value also should not be the same one specified for the `decimalChar` configuration variable, as this would create confusion when numbers were specified.

TeraScript Server uses this value in order to properly evaluate numbers in conditional comparisons (for example, Branch action, `<@IF>`, `<@IFEQUAL>` and `<@ISNUM>` Meta Tags) and in calculations performed with the `<@CALC>` Meta Tag.

The setting is also used when TeraScript Server is constructing SQL for Search, Insert, Update, and Delete actions. TeraScript automatically removes the character specified by `thousandsChar` from any values specified for numeric columns. Use the `<@DSNUM>` Meta Tag to perform the same function on numbers you specify in Direct DBMS actions.

The default value of `thousandsChar` is "," (a comma).

On Macintosh, the default is the corresponding setting in the *Numbers* control panel on the server computer. You may always revert to the default setting by assigning an empty value to this configuration variable.

thousandsChar and Scope

For more information, see "staticNumericChars" on page 467.

When `staticNumericChars` has the value `true` (the default), changing the value of `thousandsChar` has no effect during the execution of an application file. Changes to `thousandsChar` in user, domain, or system scope take effect with the *next* application file execution; as a consequence, changes to `thousandsChar` in request scope have no effect.

When `staticNumericChars` has the value `false`, `thousandsChar` works with scope in the standard way.

See also

`currencyChar`	page 402
`DBDecimalChar`	page 410
`decimalChar`	page 412
`<@DSDATE>`	page 119

<@DSNUM> page 121
<@DSTIME> page 119
<@DSTIMESTAMP> page 119
staticNumericChars page 467

threadPoolSize

System scope only


Windows and Linux only.

This variable determines the number of worker threads that TeraScript Server allocates to process requests. This is the maximum number of requests that TeraScript Server tries to process simultaneously. If the number of concurrent requests reaches this limit, additional requests are queued until threads become available. Increasing this number may have a detrimental effect on hardware that cannot support the load. The default value is `20`.

timeFormat

See "dateFormat, timeFormat, timestampFormat" on page 407.

timeoutHTML

System scope only

This configuration variable points to the HTML file that is returned when a query times out in TeraScript Server.

The file is by default called `timeout.html` and resides in the configuration directory. See “A Note on Default Locations” on page 381.

See also

`queryTimeout` page 457

timestampFormat

See "dateFormat, timeFormat, timestampFormat" on page 407.

useFullPathForIncludes

System scope only

This configuration variable specifies whether TeraScript Server uses a full file path in any files that are included in a TeraScript application file or class file using the <@INCLUDE> Meta Tag. If the variable is set to `true`, then all included paths must be specified from the root of the Web server's file system.

userAgent

Valid in all scopes

The value of this configuration variable is used in the header of HTTP requests sent by TeraScript Server as a result of timed URL execution, startup and shutdown URLs, and the URL specified in `variableTimeoutTrigger`. It is also used as the default value of the `USERAGENT` attribute of the `<@URL>` Meta Tag.

The User-Agent value in HTTP requests gives the destination server information about the program (such as, name, version, and platform) that is requesting the URL. For instance, the User-Agent value passed by Netscape Navigator 4.04 for Windows NT is:

```
Mozilla/4.04 [en] (WinNT; I)
```

Servers often use the user-agent information to determine the format of the results returned. (TeraScript application files can get the user-agent information from a request using `<@CGIPARAM NAME="USER_AGENT">`.) For example, a server may return a special version of a Web page, including Web browser-specific HTML for additional features, when the Web browser is Netscape Navigator or Internet Explorer.

The default value of `userAgent` is empty, causing TeraScript Server URL requests to use `"TeraScript Server/[version]([platform])"`, where `[version]` and `[platform]` are the values returned by `<@VERSION>` and `<@PLATFORM>` Meta Tags.

See also

`<@URL>` page 276

userKey, altuserKey

Request, application, domain and system scopes

Meta Tags evaluated

These variables set the key used to identify users in TeraScript.

The value for this configuration variable may contain Meta Tags. The tags are substituted each time the variable is used by TeraScript Server.

User variables let you store values associated with a particular user of your Web site. These values can then be accessed in any application file. In order for user variables to work properly, TeraScript must be able to uniquely identify each user who accesses it. The World Wide Web and the protocol it uses (HTTP) do not make this easy.

TeraScript gives you several options for specifying how TeraScript identifies each user. You need to choose the one that best suits your environment. You make this choice by assigning values to these configuration variables.

The `userKey` and `altuserKey` configuration variables tell TeraScript Server what piece(s) of information to use to identify a user when assigning to and evaluating user variables. The value of `userKey` is the default key for user variables. If its contents evaluate to empty, `altUserKey` is used instead.

Note If `userKey` contains a literal value, <@USERREFERENCE>, or any other Meta Tag guaranteed to return a value, then the value of `altUserKey` is irrelevant, as `userKey` will never be empty.

When you assign a value to `userKey` and `altUserKey`, you must tell TeraScript Server not to evaluate the content of the `VALUE` attribute, but instead to evaluate the Meta Tag when user variables need to be keyed. This is done with the <@LITERAL> Meta Tag.

For more information, see "<@URL>" on page 276.

The syntax of the assignment to `userKey` of its default value would be as follows:

```
<@ASSIGN NAME=userKey VALUE=<@LITERAL
VALUE="<@APPKEY><@USERREFERENCE><@CGIPARAM
CLIENT_IP>">
```

When you use <@VAR> to get the value of either of these configuration variables, the Meta Tags assigned to it are returned, not the values of those Meta Tags, because of the use of the <@LITERAL> Meta Tag. To get the actual value of the key, use the `ENCODING=METAHTML` formatting parameter in <@VAR>.

For more information, see "Encoding Attribute" on page 8.

For example, <@VAR NAME=userKey> might return <@APPKEY><@USERREFERENCE><@CGIPARAM CLIENT_IP>, indicating that user configuration variables are keyed on the TeraScript application, the TeraScript user reference ID assigned to each user, and the IP address the application file is being sent to. To get the actual value of the key for the current user, you would use <@VAR NAME=userKey ENCODING=METAHTML>, which would return the value of the string currently being used as the user key in the current application file (a 24-digit hexadecimal string).

The default value of `userKey` is <@APPKEY><@USERREFERENCE><@CGIPARAM CLIENT_IP>. The presence of the client IP address in the `userKey` ensures that a session cannot be "taken over" by someone from another IP address. The presence of <@APPKEY> in the key means that the same variable name can be used in different applications without conflicting.

The default value of `altUserKey` is empty.

See also

<@APPKEY>	page 24
<@CGIPARAM>	page 57
<@USERREFERENCE>	page 283
<@VAR>	page 286

validHosts

System scope only

This configuration variable specifies a list of hosts from which TeraScript Server accepts TeraScript CGI connections. The hosts are given in a colon-separated list, in either domain name or IP address form. This prevents an arbitrary user on your network or the Internet from using your TeraScript Server.

Any changes made to this configuration variable will have an immediate effect on the TeraScript Server.

varCachePath

System scope only

This specifies a directory to which TeraScript writes all variables when it is shutdown, and re-reads those variables from when TeraScript is started.

Note Variables continue to expire in the usual fashion; if you restart TeraScript after the specified user timeout period has elapsed, all the variables immediately expire upon being reloaded.

The default value of this variable is `{defaultpath}variables.{name of TeraScript Server}`; for example, on Windows when running TeraScript, the value of the variable is:

On Windows:

```
TERASCRIPT_PATH\Configuration\variables.Witango_Server.
```

On Unix:

```
TERASCRIPT_PATH/configuration/
variables.TeraScript_Server.
```

variableTimeout

Available in user, custom, domain, application, and system scopes

The system scope version of this configuration variable determines the default period, in minutes, after which domain and user variables expire. For user variables, the expiry timer is reset to zero each time the user accesses TeraScript Server. For application variables, the expiry timer is reset each time the TeraScript application is accessed. For domain variables, the expiry timer is reset each time a user from the domain accesses TeraScript Server. For custom variables, the expiry timer is reset each time a variable in the custom scope is accessed.

Setting `variableTimeout` to zero indicates that variables never expire. In general, this value is appropriate for the domain scope only.

To change the expiry timeout period for domain variables only, assign the desired value to `variableTimeout` in domain scope. For example, to specify that domain scope variables never expire, make the following assignment:

```
<@ASSIGN NAME=variableTimeout SCOPE=domain VALUE=0>
```

Setting this variable with user scope sets the expiry timeout for the current user, overriding the value in the system scope.

Setting this variable with application scope sets the expiry timeout for TeraScript application, overriding the value in the system scope.

See also

`variableTimeoutTrigger` page 483

variableTimeoutTrigger

User, and custom scopes

Just before a user's, or a custom scope's variables expire, the HTTP URL specified in that scope's `variableTimeoutTrigger` is activated. (The time after which variables expire is set in the configuration variable `variableTimeout`.) This URL could be used to execute an application file that clears the database of temporary user session data, purges the user name from a list of logged-in chat users, or many other possibilities.

There is no default timeout trigger. To have a trigger execute upon the expiry of each user's variables, you would assign the desired value to `variableTimeoutTrigger` (in `user` scope) at some point during each user's session. To set a trigger for a particular domain, you would assign to `variableTimeoutTrigger` in domain scope in an application file being accessed from that domain. To set a trigger for a particular application, you would assign to `variableTimeoutTrigger` in application scope in an application file being accessed from that TeraScript application.

The URL in this configuration variable cannot contain Meta Tags because the trigger mechanism does not evaluate Meta Tags. Nevertheless, you can include user-, application-, or domain-specific information in the URL by including Meta Tags in the assignment to `variableTimeoutTrigger`, which are evaluated at the time of the assignment.

See also

`mailDefaultFrom`	page 443
`userAgent`	page 477
`variableTimeout`	page 482

webServices

System scope only

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

The default value of this configuration variable is FALSE.

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 configured in the `witango.ini` file and the TeraScript 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.

See also

`webServices` page 484

webServicesExtns

System scope only

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 configured in the `witango.ini` file and the TeraScript 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.

See also

`webServices` page 484

CHAPTER 7

TeraScript Server Error Codes

7

A Listing of TeraScript Server Error Numbers and Messages

During execution of an application file by TeraScript Server, you may encounter certain error conditions. This chapter lists the main error numbers and messages generated by TeraScript Server for the various error conditions.

Note The error numbers apply only if the type of error is internal. For other types of errors (for example, DBMS), consult the appropriate documentation (for example, database driver or server).

Main Error Number	Message
-1	There was not enough memory to complete the requested operation.
-2	The specified object was not found.
-3	The application file was either missing or invalid.
-4	Unable to connect to the specified data source. Verify that data source is properly configured and that database server is online.
-5	Bad application file format version.
-8	Invalid value specified. Previous value has been used.
-9	Invalid or empty variable key.
-10	Invalid or empty variable name.
-11	Invalid or empty array name.
-12	Missing or invalid rows loop.
-13	Cannot initialize the JavaScript runtime.
-14	Invalid scope for this script.
-15	Script execution timeout.
-16	Begin Transaction encountered while existing transaction still open.
-17	End Transaction encountered with no open transaction.
-18	Error during expression evaluation.
-19	The maximum number of concurrent requests has been exceeded.
-20	The application file tag nesting exceeded limit.
-21	Loop execution timeout.
-22	The specified script language is not supported.
-23	Perl interpreter detected a syntax or runtime error.
-24	Expression Format detected a syntax or run-time error.
-25	Administrator has disabled custom scopes.
-26	TeraScript application file not part of specified application scope.
-27	Invalid Password detected.

Main Error Number	Message
-100	Data source client library not found.
-101	General error during data source operation.
-102	No data source connection exists.
-103	Connection to data source already exists.
-104	No data found.
-105	Invalid column number specified. Column number not in range of selected columns.
-106	The maximum number of concurrent connections for this data source has been exceeded. Please try again later.
-107	Could not open specified database. Verify database name and ensure proper access privileges.
-108	Maximum licensed number of connections exceeded. Please try again later.
-109	This type of data source is not supported by the server license.
-110	The specified data source cannot be found.
-111	Invalid Meta Tag for this action. A data source is needed here.
-112	Data source timed out. The data source operation took too long to execute. Try adjusting the server timeout parameter.
-113	Unable to communicate with the specified data source. The existing connection was lost. Please try again.
-114	The file specified is not part of the application designated by your server license.
-116	Could not write to configuration file. Check the file permissions.
-117	This action requires a data source.
-118	This action could not be completed because the SQL statement is too long.
-119	Invalid user name or password. Logon denied.
-201	Failed to connect to gateway.
-202	There are no gateways currently defined.
-203	A gateway with that name already exists.
-204	Failed to remove gateway.
-205	A data source with that name already exists.

Main Error Number	Message
-206	There are no data sources currently defined.
-301	The specified file or directory does not exist.
-302	A permissions error occurred while trying to access the specified file.
-303	Can not open the specified file.
-304	Unable to obtain a write lock on the file.
-305	The specified file already exists.
-306	No file name is specified.
-307	File Actions require a full file path.
-321	Unable to connect to the specified SMTP server.
-322	Mail messages must have a From address.
-323	TeraScript supports only US-ASCII (7-bit) characters in message content.
-324	Connected to SMTP server, but a communication error occurred.
-325	Mail messages must have a destination address.
-326	An invalid From address was specified.
-327	An unexpected error occurred while sending the mail message.
-328	An invalid attachment path was specified.
-329	The custom header for E-MAIL was greater than 32K.
-401	This feature has been disabled by the administrator.
-501	Only a branch to a top-level action is allowed when branching to different file.
-502	The number of nested returning Branch actions or class file method calls exceeds the limit. Check for an endless loop, or increase the returnDepth configuration variable value.
-503	The file was not loaded because it has a corrupt structure.
-504	The application file specified in this Branch action cannot be found.
-505	The Branch destination action cannot be found in the specified application file.

Main Error Number	Message
-506	The number of actions executed so far exceeds the limit. Check for an endless loop or increase the maxActions configuration variable value.
-507	TeraScript class files may not be called directly. To call a method in a TeraScript class file, use the Call Method Action instead.
-510	The structure of the application file is corrupt.
-511	This action may not have a child.
-512	This action has a malformed structure.
-513	The branch destination cannot be found or is at an invalid level.
-514	The Break action is valid only within For and While Loop actions and in groups.
-515	The Elseif/Else action is valid only when preceded by an If action.
-520	A NULL node was encountered.
-521	An empty node was encountered.
-522	A container-type node was expected but not found.
-601	System scope is for configuration variables only. Try using Domain scope instead.
-602	The Domain scope is not defined. Set the value of domainScopeKey.
-603	The array subscript is not within the range for the defined array.
-604	The string used to construct the array is invalid. Check the delimiters, and make sure the dimensions match.
-605	You cannot apply array subscript operations on scalar variables.
-606	You do not have the correct password to set system variables.
-607	You cannot get the value of the configuration password.
-608	You cannot purge the contents of the system scope.
-609	You may not set configuration variables in the cookie scope. Try using the user or local scope instead.
-610	Destination's dimensions do not match the source's for array section assignment.
-611	Row and column dimensions within the <@ARRAY> tag must be greater than 0 if there is no initialization string.

Main Error Number	Message
-612	The initialization string does not match the specified row and column dimensions, or the row and column dimensions are inconsistent within the initialization string.
-613	The row and column delimiters must be fully unique if the array dimensions are not specified.
-614	An array was expected as a parameter.
-615	Parameter arrays must have the same number of columns.
-616	A scalar cannot be pushed into an array with multiple columns.
-617	A value to add must be specified.
-618	The COLS argument cannot be parsed.
-619	The EXPR argument of a FILTER tag cannot be parsed.
-620	Invalid scope specified. This scope is valid only in methods.
-621	The specified variable is read only.
-622	The specified object instance could not be created.
-623	You do not have the correct password to purge cache.
-624	The application scope is not defined.
-625	The POSTARGARRAY argument of the URL tag requires an array with exactly two columns.
-626	Invalid Object type, must be one of: COM, JAVABEAN, or TCF.
-700	Invalid outer join.
-701	A table specified is an inner table to more than one outer table.
-702	Outer join specified creates cyclic join relationships.
-800	An error occurred while preparing the parameters for this method invocation.
-801	An error occurred while invoking this method.
-802	An error occurred while processing the results of this method invocation.
-803	The specified object's handler doesn't support method invocation.
-804	The specified method is not implemented.
-805	Insufficient security for the requested operation.
-806	An unspecified method call error occurred.

Main Error Number	Message
-807	Cannot access handler.
-808	Cannot create native object.
-809	Cannot bind to native object.
-810	Error getting object's introspection info.
-811	Given buffer is too small.
-812	A memory error occurred.
-813	A bad state transition was attempted.
-814	Bad or missing object identity.
-815	Cannot locate object.
-816	The requested data is not available.
-817	The object does not hold an open collection.
-818	No license to use this object.
-819	The native object was released.
-820	Requested feature not implemented by handler.
-821	Unspecified error.
-822	The specified collection index is not valid for this object.
-901	The specified object is not a document.
-902	An error occurred while parsing the XML.
-903	The specified element cannot be found, or element specifier is empty.
-904	An error occurred while updating the document object or element.
-905	An error occurred while deleting the document object or element.
-906	An error occurred while creating the document object.
-1000	The maximum number of concurrent URL requests has been exceeded. Please try again later.
-1020	The Arguments associated with tags are not defined.
-1030	External action environment variable has value but no name.
-1031	External action environment variable has name but no value.

Main Error Number	Message
-1032	Your request could not be processed because this personal server is already serving requests from another IP address.
-1050	The specified Purge call could not be executed.
-1060	Maximum licensed number of user exceeded. Please try again later.

CHAPTER 8

Web Services

8

How to publish a Web Service using TeraScript

The TeraScript Server 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 TeraScript 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 TeraScript 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 at `http://www.terascript.com` to create your WSDL file.
- Configuring the TeraScript Server so that it knows which WSDL files are linked to which `tcf` files.

Note TeraScript'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)

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

TeraScript 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. TeraScript has a WSDL generator available on http://www.terascript.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 501.

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 andf 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 TeraScript 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 TeraScript 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 TeraScript 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.

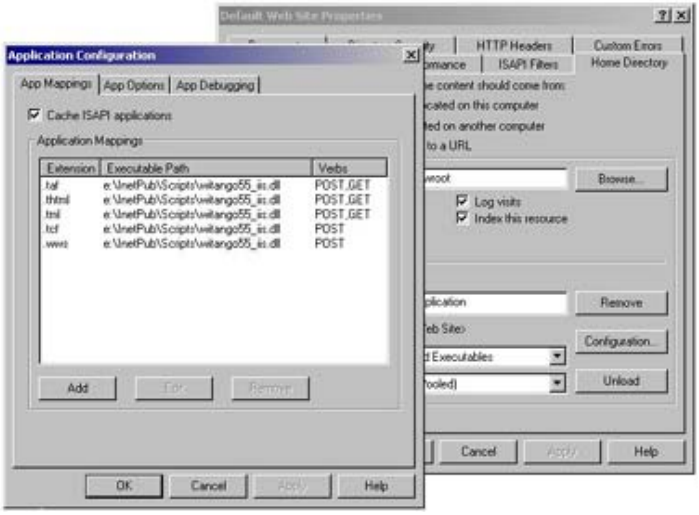
Note These Web Services parameters can be controlled using the TeraScript Administration Application.

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 TeraScript 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.

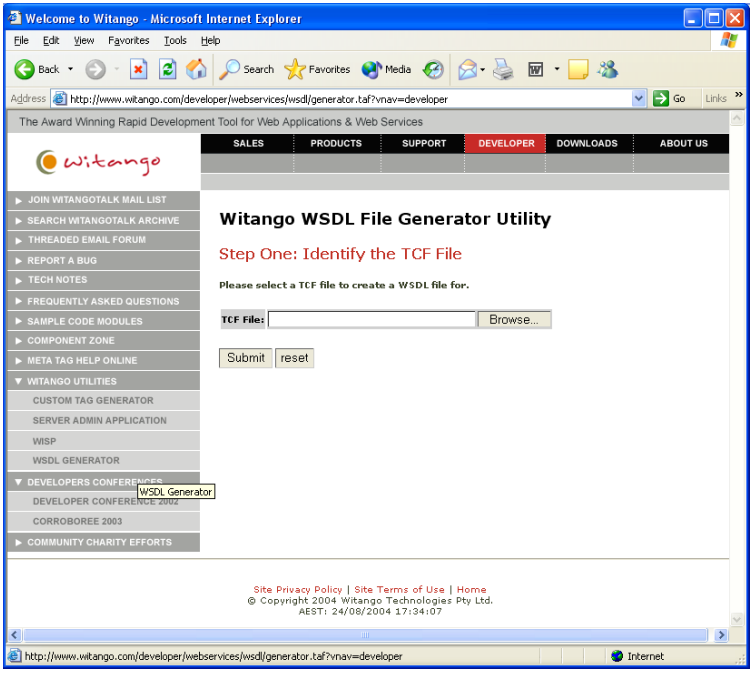
Creating a WSDL file for your tcf

A utility has been written that will automatically create a WSDL file from a tcf. This utility is available in the Developer section at http://www.terascript.com/developer.

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.

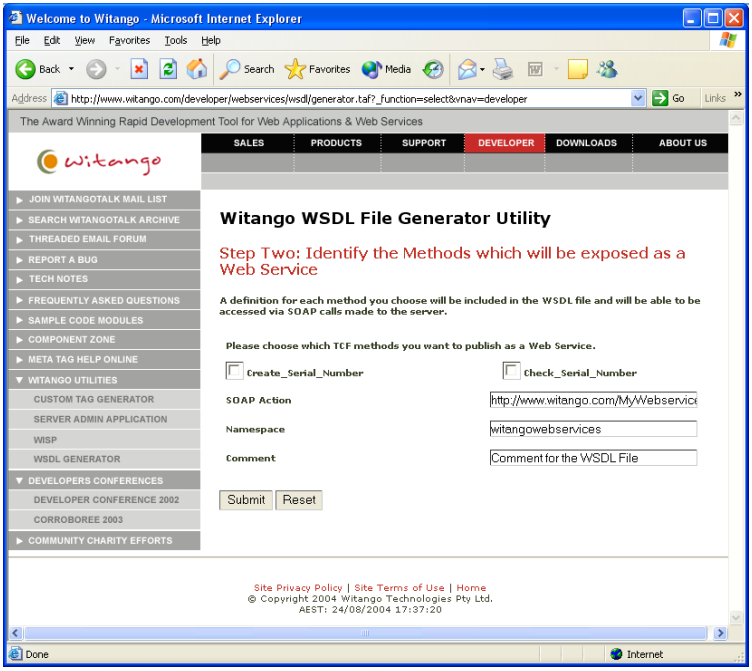
Steps to Create your WSDL file

1. Go to the TeraScript WDSL File Generator Utility at `http://www.TeraScript.com`
2. The Form shown below will appear.



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.

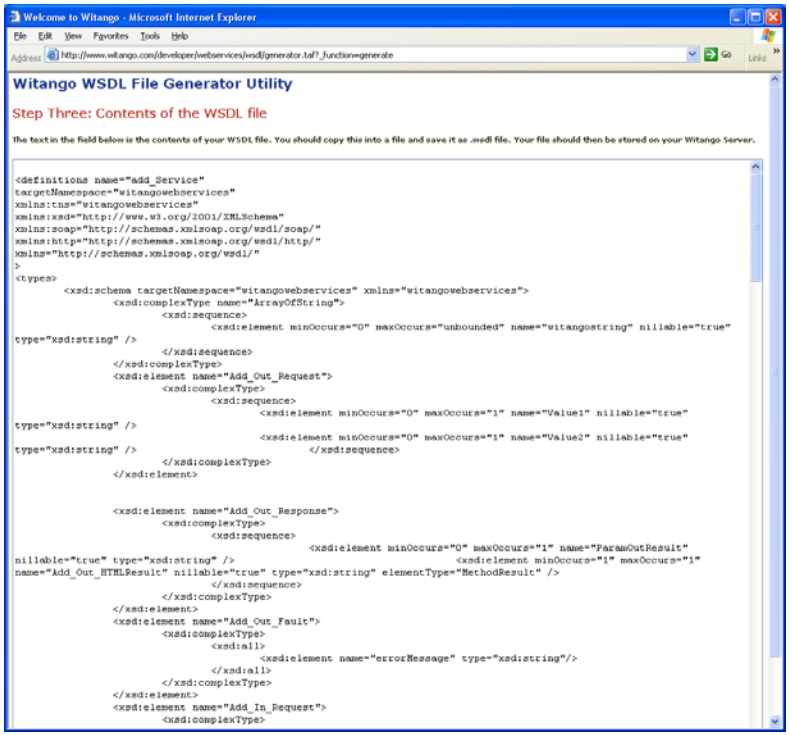
6 Set the **SOAP Action** allows the user to indicate the intent of the SOAP HTTP request. The value is a URI identifying the intent. No value means that there is no indication of the intent of the message. Basic TeraScript Web Services do not require this field - so it can be left as an empty string.

7 Set the **NameSpace** allows the user to provide a namespace to avoid a conflict with element names.

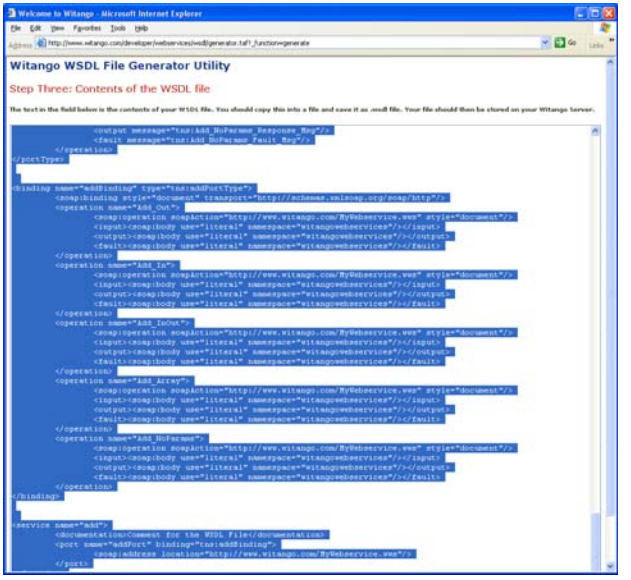
8 Set the **Comment** this is optional and can be used to add a comment to the WSDL file.

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

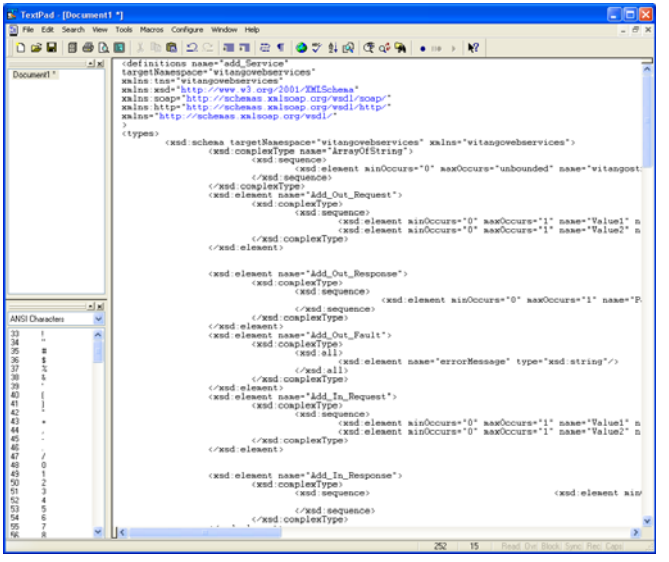
Note The contents of the WSDL file is typically very large and you will need to scroll through the entire field to see it all.



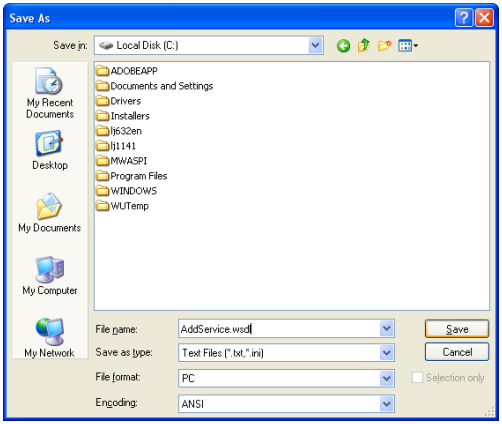
10 **Highlight** ALL of the text and **COPY** it to your clipboard.



11 **Open** a Text Editor and **PASTE** the text into a new document.



12 **Save** the new document as a `<webservicename>.wdsl` file.



13 Move the WSDL file to your TeraScript Server.

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

Configuring the TeraScript 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 `WDSL` 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 TeraScript 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`

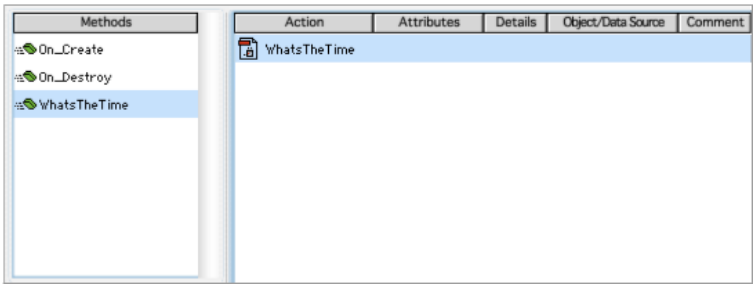
Is automatically populated with the body of the SOAP request.

How to consume your TeraScript Web Service

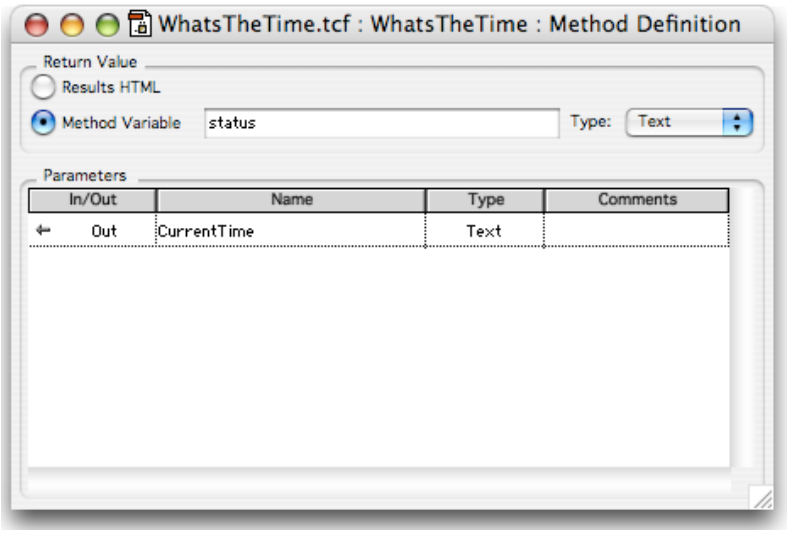
Creating a Simple Web Service

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

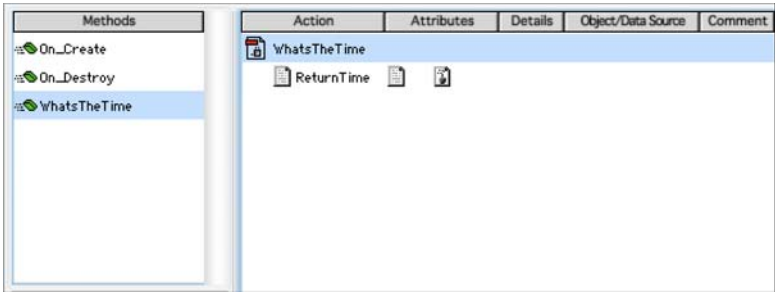
1. Create new tcf and save it as WhatsTheTime.tcf in the `<webroot>/webservices/WhatsTheTime` directory
2. Add a new method and call it `WhatsTheTime`



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

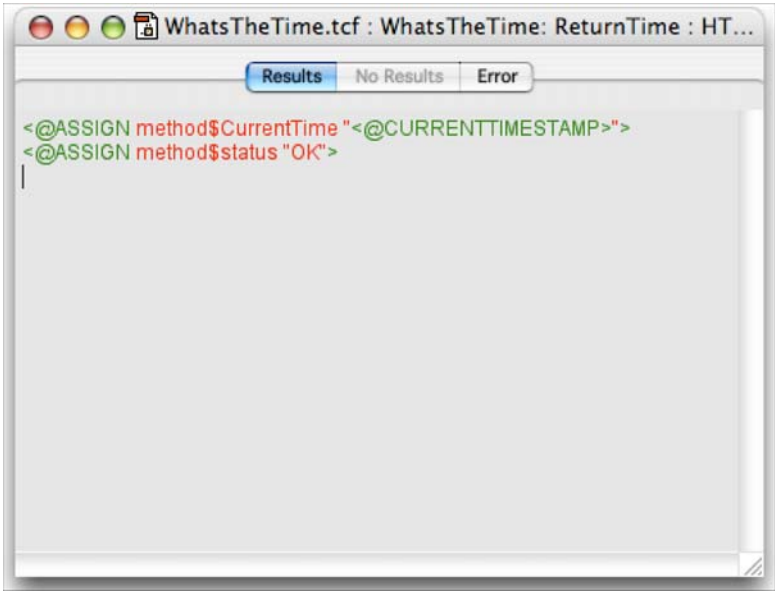


4 Add a Result action to the method and name it `ReturnTime`.

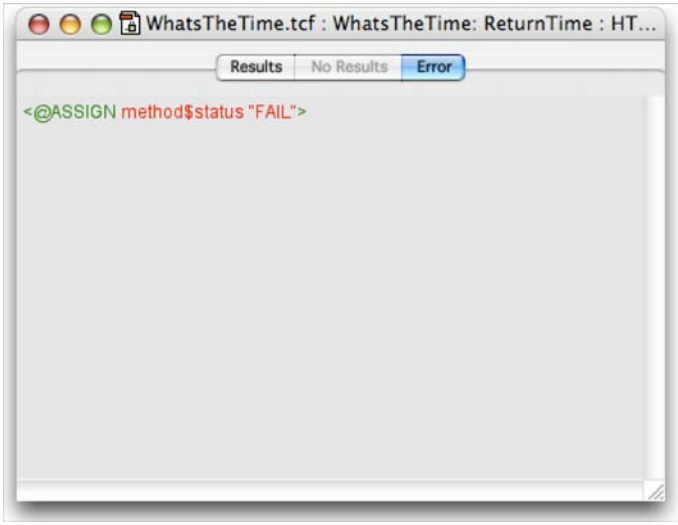


5 In the Results tab add the following code:

```
<@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 501

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="witangowebservices"
xmlns:tns="witangowebservices"
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="witangowebservices"
xmlns="witangowebservices">
<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 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="witangowebservices"/></input>
<output><soap:body use="literal"
namespace="witangowebservices"/></output>
<fault><soap:body use="literal"
namespace="witangowebservices"/></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 499.

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

```
[webservices]
WhatsTheTime=

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

10 Restart the TeraScript 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="witangowebservices" />
```

```
</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: TeraScript Server
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="witangowebservices" />
</env:Body>
</env:Envelope>
```

6 The web service will responds 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="witangowebservices">
        <CurrentTime>12/08/2004 11:46:49</CurrentTime>
       <status></status>
 </mns:WhatsTheTime_Response>
</env:Body>
</env:Envelope>
```

Calling the web service via a TeraScript File

Applying the above example to a TeraScript 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="witangowebservices" />

</env:Body>

</env:Envelope>

'>

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

<@URL

LOCATION='@@request$WebService'

USERAGENT='TeraScript <@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>
```

CHAPTER 9

<@CALC> Expression Operators

9

A List of Expression Operators for use with the <@CALC> Meta Tag

This chapter covers the following topics:

- numbers
- hexadecimal, octal and binary numbers
- arithmetic operators
- mathematical functions
- string functions
- logical operations
- comparison operations
- calculation variables
- sub-expressions

Numbers

A valid number for use in the <@CALC> Meta Tag is a sequence of digits, optionally preceded or trailed by a currency sign (default "$", otherwise set by the configuration variable `currencyChar`), with any number of thousand separator characters, an optional decimal point, and an exponentiation part. As well, an empty variable or empty string evaluates to zero.

Numbers can be used with any operators and functions, even with the string specific function *len*, which returns the length of the number converted to a string.

When a number is used in logical expression, any non-zero number is considered *true*, and zero is considered *false*.

Logical expressions themselves return "1" if they are *true* or "0" if they are *false*.

Two symbolic constants, `true` and `false`, which evaluate to "1" and "0", respectively, are provided for convenience.

An empty string evaluates to zero for the purposes of calculation. That is, if the variable `foo` is empty, the following operations are valid:

```
<@CALC '@@foo + 1'>        OK, returns 1
<@CALC '"" + 1'>           OK, returns 1
<@CALC 'mean(@@foo 1)'>    OK, returns 0.5
```

The thousand separator set to space

A special case occurs when the thousand separator is set to a space. A number containing a space can be processed if it is a result of a tag evaluation; however, a number literal must be quoted if it includes spaces.

For example:

```
<@ASSIGN NAME=fred VALUE="1 000 000">

<@CALC "@@fred / 100">     Ok, returns 10000.0
<@CALC "@@fred > '1 000'"> Ok, returns 1.0
<@CALC "@@fred > 1 000">   Error
```

For more information, see “currencyChar” on page 396, decimalChar on page 406, DBDecimalChar on page 404, and thousandsChar on page 465.

The thousands separator, currency sign, and other numerical formats are set by TeraScript configuration variables. They can be set in various scopes.

Array evaluation

<@CALC> treats array references using non-array-specific operators and functions as a numerical value returning the number of rows in the array.

This provides an easy way to verify whether an array is empty or contains a certain value. For example, you can test for the existence of an array variable with `<@CALC EXPR="@@array_variable > 0" TRUE="Yes!" FALSE="No such variable.">`.

For example:

The variable `fred` contains the following array:

1	2
3	4

The variable `barney` contains the following array:

1	2
5	6
7	8

`<@CALC @@fred>` returns 2.

`<@CALC @@barney>` returns 3.

`<@IF EXPR="@@fred > @@barney" TRUE="true!" FALSE="alas">` returns “alas”.

Hexadecimal, Octal and Binary Numbers

The calculator can accept hexadecimal, octal, and binary numbers. The *num* function converts strings representing hexadecimal, octal and binary numbers to decimal numbers, and the result of the conversion can be used anywhere where a number is used. The following table specifies the conversion rules.

Note If a decimal number is passed to this function, it either yields an error or an incorrect result.

Prefix	Valid Symbols	Converted As	Examples
0x	0123456789abcdef	Hexadecimal	num (*0xff*) num (*0x0123f3a4*)
0	01234567	Octal	num (*0123456*) num (*0120235*)
None	01	Binary	num (*1011110010100*) num (*111*)

For example, all the following expressions generate errors:

```
num(0x123fga)    ERROR: letter g is invalid
num(012380)      ERROR: digit 8 is invalid
num(123)         ERROR: digits 2 and 3 are invalid
```

Strings

Any TeraScript Meta Tag that does not evaluate to a valid number or array reference is considered a string. No additional quoting is required. There is a single exception to this rule, further explained in Meta Tag Evaluation on page 527.

Strings can be used only in comparison operations, *contains* clauses or as arguments to the *len* function. A string literal—that is, a string, directly included in the expression—must be enclosed in single quotes if it contains spaces, special characters or starts with a digit.

Note Single letters must always be enclosed in quotes in string operations so that they are treated as letters, and not as calculation variables.

For more information, see "Calculation Variables" on page 521.

The following examples show string comparisons. If a string literal contains a single quote or a backslash, it must be escaped with a backslash.

```
<@ASSIGN NAME=name VALUE="John Lennon">
<@CALC EXPR="@@name=John"> false
<@CALC EXPR="@@name=John Lennon"> ERROR
<@CALC EXPR="@@name='John Lennon'"> true
<@CALC EXPR="@@name='John*'"> true

<@ASSIGN NAME=name VALUE="John's trousers">
<@CALC EXPR="@@name=John*" true
<@CALC EXPR="@@name='John\'s trousers'"> true
<@CALC EXPR="@@name='John's'"> ERROR

<@ASSIGN NAME=dir VALUE="C:\test">
<@CALC EXPR="@@dir='C:\test'"> false
<@CALC EXPR="@@dir='C:\\test'"> true
```

When a string is encountered on one side of the comparison operation, the other operand is forced to a string, too. For example:

```
2.15 <='abba'
'123.456.78.12'=@@ip_address
```

Function *len* returns the length of the string, so the result of this operation can be used anywhere a number can be used. Strings can not be assigned to calculation variables.

For example, these are valid expressions:

```
ABBA='BLACK SABBATH' false
len(JOHN LENNON) + len(FREDDY MERCURY) - 5 > 0 true
```

but these are not:

```
a :=ABBA        ERROR: cannot assign string
FREDDY < 0      ERROR: cannot compare string and number
```

and this tag returns true although you may expect it to return false:

```
<@CALC EXPR="a=b">
```

Note A single letter on both sides of the comparison operator evaluates to a calculation variable, meaning a number comparison is performed.

String comparisons using <@CALC> are case insensitive.

Calculation Variables

A calculation variable is a single case-insensitive letter (A–Z) that can be assigned a numeric value and used in subsequent operations. You can write small programs inside the tag with calculation variables and statement separators, or put a program in a separate file and use <@INCLUDE> to calculate the result.

Single letters must always be enclosed in quotes in string operations so that they are treated as letters, and not as calculation variables. For example:

For more information, see "beginswith" on page 523.

<@CALC EXPR="Henry beginswith 'H'"> evaluates the string "Henry" to see if it begins with the string "H" (case-insensitive).

<@CALC EXPR="1234 beginswith H"> evaluates "1234" to see if it begins with the value specified in the calculation variable H (number-to-string conversions are performed).

The following table shows predefined calculation variables. You may use these values in your programs, or have any of these calculation variables reassigned with any other value.

Variable	Meaning	Value
G	**(3 - sqrt(5))/2**, the golden ratio.	0.381966011250105
E	**e**, the base of natural logarithms.	2.718281828459045
L	**$\log_{10}$(e)**, the ratio between natural and decimal logarithms.	0.434294481903252
P	**pi**, the circumference to diameter ratio of a circle.	3.141592653589793
Q	**sqrt(2)**, the square root of 2.	1.414213562373095
I	Has a meaning only inside *foreach* expression.	Current row index
J	Has a meaning only inside *foreach* expression.	Current column index
X	Has a meaning only inside *foreach* expression.	Current array element index

Operators

The following table shows the operators listed in order of increasing precedence. Operators having the same precedence, for example, plus and minus, are not separated by a rule.

Note The `beginswith` operator should be used instead of a trailing asterisk as a wildcard in comparisons.The use of asterisks as wildcards is deprecated and will be removed in a future release.

Operator	Meaning and Return Value	Usage
;	Sub-statement separator, returns the value of the last statement.	*statement* ; *statement*
:=	Assignment operator, assigns the value of the expression to the calculation variable, and returns that value.	*variable* := *expression*
|| OR	Logical OR, returns 1 if any of the expressions is evaluated to a non-zero value, or 0 otherwise.	*expr* || *expr* *expr* OR *expr*
&& AND	Logical AND, returns 1 if both of the expressions are evaluated to non-zero values, or 0 otherwise.	*expr* && *expr* *expr* AND *expr*
<	Numeric or string LESS. Returns 1 if left operand is greater than right one, or 0 otherwise.	*expr* < *expr* *string* < *string*
>	Numeric or string GREATER. Returns 1 if left operand is greater than right one, or 0 otherwise.	*expr* > *expr* *string* > *string*
<=	Numeric or string LESS OR EQUAL. Returns 1 if left operand is less than or equal to right one, or 0 otherwise.	*expr* <= *expr* *string* <= *string*
>=	Numeric or string GREATER OR EQUAL. Returns 1 if left operand is greater than or equal to right one, or 0 otherwise.	*expr* >= *expr* *string* >= *string*
=	numeric or string EQUAL. Returns 1 if left operand is equal to right one, or 0 otherwise.	*expr* = *expr* *string* = *string*
!=	Numeric or string NOT EQUAL. Returns 1 if left operand is not equal to right one, or 0 otherwise.	*expr* != *expr* *string* != *string*
? :	Ternary comparison. Evaluates to *expr1* if condition is true, or to *expr2* otherwise.	(*cond*) ? *expr1*: *expr2*

Operator	Meaning and Return Value	Usage
contains	Containment. Returns true if specified string or number is contained in the array.	*array* contains *string* *array* contains *number*
contains	Occurrence. Returns true if specified string or number is a substring of the source string.	*source_string* contains *string* *source_string* contains *number*
beginswith	Occurrence. Returns true if specified string or number begins the source string. (Case-insensitive.)	*source_string* beginswith *string* *source_string* beginswith *number*
endswith	Occurrence. Returns true if specified string or number ends the source string. (Case-insensitive.)	*source_string* endswith *string* *source_string* endswith *number*
+	Addition. Returns the sum of the expressions.	*expr* + *expr*
–	Subtraction. Returns the difference of the expressions.	*expr* – *expr*
*	Multiplication. Returns the product of the expressions.	*expr* * *expr*
/	Division. Returns the quotient of the expr1 divided by the expr2.	*expr1* / *expr2*
%	Modulo. Returns the remainder of expr1 divided by expr2.	*expr1* % *expr2*
^	Power. Returns expr1 raised to expr2 power.	*expr1* ^ *expr2*
–	Unary minus. Returns the negation of the expression.	– *expr*
+	Unary plus. Returns the expression itself.	+ *expr*
! NOT	Logical NOT. Returns 0 if the value of the expression is not 0, or 1 otherwise.	! *expr* NOT expr

Built-in Functions

Each built-in function expects either a single numeric argument, or a space-separated list of mixed numeric and array arguments, or a string. It is an error to specify an argument of the wrong type to a function. If an array, specified as an argument to a function, contains non-numeric elements, these elements are ignored without any error diagnostics.

The following tables list all built-in functions.

Numeric functions of the form func(expr)

Function	Meaning and Return Value	Arguments and Usage
abs	**|x|**, the absolute value of the expression	abs(*expr*)
acos	$\boldsymbol{\cos^{-1}(x)}$, the arccosine of the expression, returned in radians	acos(*expr*)
asin	$\boldsymbol{\sin^{-1}(x)}$, the arcsine of the expression, returned in radians	asin(*expr*)
atan	$\boldsymbol{\tan^{-1}(x)}$, the arctangent of the expression, returned in radians	atan(*expr*)
ceil	expression rounded to the closest integer greater than or equal to the expression	ceil(*expr*)
cos	***cos(x)***, the cosine of the expression, specified in radians	cos(*expr*)
exp	$\boldsymbol{e^x}$, the exponentiation of the expression	exp(*expr*)
fac	***x!*** (or ***1*2*3*...*x***) factorial of the expression	fac(*expr*)
floor	expression rounded to the closest integer less than the expression	floor(*expr*)
log	***ln(x)*** (or $\boldsymbol{\log_e(x)}$), the natural logarithm of the expression	log(*expr*)
log10	$\boldsymbol{\log_{10}(x)}$, the decimal logarithm of the expression	log10(*expr*)
sin	***sin(x)***, the sine of the expression, specified in radians	sin(*expr*)
sqrt	***sqrt(x)*** (or $\boldsymbol{x^{1/2}}$), the square root of the expression	sqrt(*expr*)
tan	***tan(x)***, the tangent of the expression, specified in radians	tan(*expr*)

String functions of the form *func(string)*

Function	Meaning and Return Value	Arguments and Usage
len	returns the length of the string enclosed in parentheses	len(*text*)

Function	Meaning and Return Value	Arguments and Usage
num	converts a string, representing a hexadecimal, octal, or binary number into a number	num(*text*)

Array functions of the form *func(expr|array expr|array)*

Function	Meaning and Return Value	Arguments and Usage
max	$\boldsymbol{max(A_1, A_2,...A_n)}$. returns the largest element	max(*expr expr* ...)
min	$\boldsymbol{min(A_1, A_2,...A_n)}$. returns the smallest element	min(*expr expr* ...)
sum	$\boldsymbol{A_1+A_2+...+A_n}$. returns the sum of the elements	sum(*expr expr*...)
prod	$\boldsymbol{A_1*A_2*...*A_n}$. returns the product of the elements	prod(*expr expr*...)
mean	$\boldsymbol{A_{mean}=(A_1+A_2+...+A_n)/n}$. returns the mean of the elements	mean(*expr expr*...)
var	$A_{var}=((A_1-A_{mean})^2+((A_2-A_{mean})^2+...+((A_n-A_{mean})^2)/(n-1)$ returns the (squared) variance of the elements	var(*expr expr*...)

Array Operators

Contains Operator

The *contains* operator has the following syntax:

```
<@VAR NAME="array"> contains number or string
```

This operator checks if the specified number or string is contained in the array. The string should be enclosed in quotes, if it contains any non-alphanumeric characters. The operator returns "1" if the element is found, or "0" otherwise.

For example, the following expression, which uses the <@IF> Meta Tag, returns "Cool" if "Queen" is found in the CDs array, and "Too Bad" if it is not.

```
<@IF EXPR="<@VAR NAME=CDs> contains Queen" TRUE=Cool
FALSE="Too bad">
```

Foreach Operator

The *foreach* operator has the following syntax:

```
<@VAR array> foreach {statement; ...}
```

For more information, see "<@ARRAY>" on page 29.

This operator steps through the elements of an array and it assigns

- the value of the elements to the variable "X"
- the current row number to the variable "I"
- and the current column number to the variable "J"

For more information, see "<@ASSIGN>" on page 32.

and it executes the statements inside the braces "{ }"for each element. All non-numeric elements are interpreted as zeroes.

The operator returns the last calculated value of the expression.

The values of "X, I, J" are restored upon the exit from the foreach operator. For example, if array CDs is initialized as follows:

```
<@ASSIGN NAME="CDinitValue" VALUE="AC/
DC,Scorpions,Deep Purple,Black
Sabbath,Queen;19.50,22.50,22.50,17.90,29.00">
<@ASSIGN NAME="CDs" VALUE="<@ARRAY ROWS='2'
COLS='5' VALUE=@@CDinitValue CDELIM=','
RDELIM=';'>">
```

then the following program prints the name of the most expensive CD:

```
<@VAR NAME=CDs[1,<@CALC "t :=1; p :=0.0;

<@VAR NAME=CDs> foreach
{ t :=(p < x)? j: t; p :=(p < x)? x: p; }; t">]>
```

Meta Tag Evaluation

There are two special cases when a Meta Tag is not treated as a string. Consider the following two examples:

```
<@CALC EXPR="<@POSTARG NAME=prog>">
<@CALC EXPR="<@INCLUDE FILE=myprog>">
```

If the post argument *prog* contains an expression submitted by a user, or the file *myprog* contains an expression to be calculated, one would expect <@CALC> to produce the result of the calculation. The rule is, if the expression contains a single Meta Tag, such an expression is fully evaluated by the calculator, rather than treated as a string.

Ordering of Operation Evaluation With Parentheses

Parentheses can be used to order the evaluation of expressions that otherwise are evaluated in the order specified in the Operators table (page 522). For example:

```
<@CALC EXPR= "7*3+2">
```

This example evaluates to “23”.

```
<@CALC EXPR= "7*(3+2)">
```

This example evaluates to “35”.

A more complex example can be constructed using different operators and nested parentheses:

```
<@CALC EXPR="(<@ARG _function> = 'detail') and
((len(<@ARG id>) != 0 and <@ARG mode>='abs')
or (<@ARG mode>='next' or <@ARG mode>='prev'">))>
```

This tag evaluates to “1” (true) if the `_function` argument is equal to “detail” *and* any one of the following conditions are met:

- `id` arg is not empty *and* the `mode` arg is “abs”
- `mode` argument is “next”
- `mode` argument is “prev”.

See Also

`<@CALC>` page 41

CHAPTER 10

Lists of Meta Tags

10

A listing of TeraScript Server Meta Tags

This Appendix displays Meta Tags in the following listings:

- alphabetical table of Meta Tags and Meta Tags with their attributes
- alphabetical listing of Meta Tags by function
- alphabetical reference to all Meta Tags, their function, syntax and explanation.

Alphabetical List of Meta Tags

Meta Tag	Abstract
<@ABSROW>	Returns the position of the current row within the total rowset.
<@ACTIONRESULT>	Returns the value of the specified item from the first row of results of the specified action.
<@ADDROWS>	Adds one or more rows to an array.
<@APPFILE>	Returns the path to the current application file, including the file name.
<@APPFILENAME>	Returns the current application file's name.
<@APPFILEPATH>	Returns the path to the current application file, excluding the application file name, but including the trailing slash.
<@APPKEY>	Returns the key value of the current application scope.
<@APPNAME>	Returns the name of the current application.
<@APPPATH>	Returns the path to the current application
<@ARG>	Returns search and/or post argument values.
<@ARGNAMES>	Returns an array of all search and post arguments passed to the current application file.
<@ARRAY>	Returns an array with a specified number of rows and columns.
<@ASCII>	Returns the ASCII value of the first character in a string.
<@ASSIGN>	Assigns a value to a variable.
<@BIND>	Explicitly passes a value in the Direct DBMS action using the parameter binding capabilities of ODBC or OCI.
<@BREAK>	Ends execution of a loop.
<@CALC>	Returns the result of a calculation.
<@CALLMETHOD>	Calls a specified method of an object
<@CGI>	Returns the full path and name of the TeraScript CGI.

<@CHAR>	Returns the character that has the specified ASCII value.
<@CHOICELIST>	Creates HTML selection list boxes, pop-up menus/drop-down lists, and radio button clusters using data from variables, database values, and so on.
<@CIPHER>	Performs encryption/decryption on strings.
<@CLASSFILE>	Returns the path to the current TeraScript class file, including the file name.
<@CLASSFILEPATH>	Returns the path to the current TeraScript class file, excluding the TeraScript class file name.
<@CLEARERRORS>	Clears errors and allows TeraScript Server to resume processing..
<@COL>	Returns the value of a numbered column.
<@COLS> </@COLS>	Processes the enclosed HTML once for each column in the current row.
<@COLUMN>	Returns the value of a named column.
<@COMMENT> </ @COMMENT>	Includes comments in TeraScript application files.
<@CONFIGPATH>	Returns the full path to the configuration directory of TeraScript Server.
<@CONNECTIONS>	Provides information about each data source, mail server, or external action currently in use by TeraScript Server.
<@CONTINUE>	Ends the current iteration of a loop.
<@CREATEOBJECT>	Creates a new instance of a particular object.
<@CRLF>	Evaluates to a carriage return/linefeed combination. Used in the file pointed to by headerFile (the HTTP header).
<@CURCOL>	Returns the index (1, 2, 3...) of the column currently being processed if placed inside a <@COLS></@COLS> block.
<@CURRENTACTION>	Returns the name of the executing action.
<@CURRENTDATE>, <@CURRENTTIME>, <@CURRENTTIMESTAMP>	Returns the current date, time, or timestamp.
<@CURROW>	Returns the number of the current row being processed in a <@ROWS> or <@FOR> block.

<@CUSTOMTAGS>	Returns an array of all custom Meta Tags in the scope specified.
<@DATASOURCESTATUS>	Returns an array containing summary information about data sources, mail servers or external actions used by TeraScript Server.
<@DATEDIFF>	Returns the number of days between the two dates specified.
<@DATETOSECS>, <@SECSTODATE>	Converts a date into seconds.
<@DAYS>	Adds days to a date.
<@DBMS>	Returns the concatenated name and version of the database used by the current action's data source.
<@DEBUG> </@DEBUG>	Delimits text to appear in Results HTML only in debug mode.
<@DEFINE>	Creates an empty variable.
<@DELROWS>	Deletes one or more rows from an array.
<@DISTINCT>	Returns an array containing the distinct rows in the input array.
<@DOCS>	Displays the content of an application file in HTML.
<@DOM>	Parses XML into a document instance.
<@DOMAIN>	Returns the key value of the current domain scope.
<@DOMDELETE>	Deletes XML from a document instance.
<@DOMINSERT>	Inserts XML into a document instance.
<@DOMREPLACE>	Replaces XML in a document instance.
<@DOMSEARCH>	Searches a DOM.
<@DQ>, <@SQ>	Returns a double quote, for use within quoted attributes.
<@DSDATE>, <@DSTIME>, <@DSTIMESTAMP>	Converts a date, time, or timestamp value to the format required by the current action's data source.
<@DSNUM>	Converts a number to the format required by the current action's data source.
<@DSTYPE>	Returns the type of data source associated with the current action.
<@ELEMENTATTRIBUTE>	Returns the value of one or more attributes from a document instance.

<@ELEMENTATTRIBUTES>	Returns the value of all attributes of one or more elements from a document instance.
<@ELEMENTNAME>	Returns an element name or names from a document instance.
<@ELEMENTVALUE>	Returns an element value or values from a document instance.
<@EMAIL>	Enables the composition and manipulation of an email message.
<@EMAILSESSION>	Enables the sending and receiving of email messages using the email protocols SMTP, POP3 and IMAP4.
<@ERROR>	Returns the value of the named error component of the current error.
<@ERRORS> </@ERRORS>	In conjunction with <@ERROR>, iterates over a list of errors.
<@EXCLUDE> </@EXCLUDE>	Processes text for Meta Tags, without adding the results of that processing to the Results HTML.
<@EXIT>	Ends the processing of current HTML and continues with the next action in the application file.
<@FILTER>	Returns an array containing rows matching a specified expression.
<@FOR> </@FOR>	Allows looping in HTML.
<@FORMAT>	Allows formatting of text, numeric, and datetime values.
<@GETPARAM>	Retrieves the contents of a parameter variable within a TeraScript class file.
<@HTTPATTRIBUTE>	For manipulation of default headers.
<@HTTPSTATUSCODE>	For manipulation of default headers.
<@IF>, <@ELSEIF>, <@ELSEIFEMPTY>, <@ELSEIFNOTEMPTY>,<@ELSEIFEQUAL>, </@IF>	Performs conditional processing.
<@IFEMPTY> <@ELSE> </@IF>	Includes text in HTML if a provided value is empty.
<@IFEQUAL> <@ELSE> </@IF>	Includes text in HTML if two values are equal.
<@IFNOTEMPTY> <@ELSE> </@IF>	Includes text in HTML if a provided value is not empty.
<@INCLUDE>	Returns the contents of a specified file.
<@INTERSECT>	Returns the intersection of two arrays.

<@ISALPHA>	Checks whether a value is a valid string containing only alphabetical characters.
<@ISDATE>, <@ISTIME>, <@ISTIMESTAMP>	Checks whether a value is a valid date, time, or timestamp.
<@ISMETASTACKTRACE>	Checks whether a metastacktrace is available.
<@ISNULLOBJECT>	Tests whether a variable is a null object.
<@ISNUM>	Checks whether a value is a valid number.
<@KEEP>	Returns a string stripped of specified characters.
<@LDAPADD>	Adds a node to an LDAP directory server
<@LDAPDELETE>	Deletes a node from an LDAP directory server
<@LDAPMODIFY>	Modifies a node on an LDAP directory server
<@LDAPSEARCH>	Searches for a node on an LDAP directory server
<@LEFT>	Returns the first *n* characters from a string.
<@LENGTH>	Returns the number of characters in a string.
<@LITERAL>	Causes TeraScript to suppress Meta Tag substitution for the supplied value.
<@LOCATE>	Returns the starting position of a substring in a string.
<@LOGMESSAGE>	Saves a message to the TeraScript Server log file.
<@LOWER>	Converts a string to lowercase.
<@LTRIM>	Returns string stripped of leading spaces.
<@MAKEPATH>	Performs normalisation of paths.
<@MAP>	Concatenates columns of an array.
<@MAXROWS>	Returns the value specified in the Maximum Matches field of a Search or Direct DBMS action.
<@METAOBJECTHANDLERS>	Returns an array with a row for each object-handling plug-in.
<@METASTACKTRACE>	Returns an array containing the Meta Stack Trace.
<@MIMEBOUNDARY>	Generates a MIME boundary string.
<@NEXTVAL>	Increments a variable and returns the value.
<@NULLOBJECT>	Creates null objects.

<@NUMAFFECTED>	Returns the number of rows affected by the last executed Insert, Update, Delete, or DirectDBMS action.
<@NUMCOLS>	Returns the number of columns retrieved by an action or in a specified array.
<@NUMOBJECTS>	Returns the count of the objects in the collection or iterator.
<@NUMROWS>	Returns the number of rows retrieved by an action or in a specified array.
<@OBJECTAT>	Given an iterator or collection object and an index, returns a single item from the object.
<@OBJECTS></@OBJECTS>	Loops through collection and iterator objects in variables returned by method calls.
<@OMIT>	Returns a string stripped of specified characters.
<@PAD>	Returns a padded string appending or prefixing a given character.
<@PLATFORM>	Returns the name of the operating platform.
<@POSTARG>	Returns the value of the named post argument.
<@POSTARGNAMES>	Returns an array containing the names of all post arguments.
<@PRODUCT>	Returns the name of TeraScript Server's product type.
<@PURGE>	Removes one or all variables from a scope.
<@PURGECACHE>	Allows selective purging of the file cache.
<@PURGEDEBUG>	Empties the accumulated debug output up to the point that the tag is executed.
<@PURGERESULTS>	Empties the accumulated Results HTML.
<@RANDOM>	Returns a random number.
<@REGEX>	Finds strings using regular expressions.
<@RELOADCONFIG>	Forces a reload of configuration files.
<@RELOADCUSTOMTAGS>	Forces a reload of the custom tags file of the specified scope.
<@REPLACE>	Replaces strings.
<@RESULTS>	Evaluates to the accumulated Results HTML.
<@RIGHT>	Extracts the last *n* characters from the string.

<@ROWS> </@ROWS>	Allows iteration over the rows of an action's results or an array.
<@RTRIM>	Returns a string stripped of trailing spaces.
<@SCRIPT>	Executes scripts written in JavaScript.
<@SEARCHARG>	Returns the value of the specified search argument.
<@SEARCHARGNAMES>	Returns an array containing the names of all search arguments.
<@SECSTODATE>, <@SECSTOTIME>, <@SECSTOTS>	Converts seconds to a date. Converts seconds to a time. Converts seconds to a timestamp.
<@SERVERNAME>	Returns the name of the current TeraScript server.
<@SERVERSTATUS>	Returns status information about TeraScript Server.
<@SETCOOKIES>	Returns the correct Set-Cookie lines to set the values of cookie variables.
<@SETPARAM>	Sets the value of a parameter variable within a TeraScript class file.
<@SLEEP>	Sleeps a worker thread.
<@SORT>	Sorts the input array by the column(s) specified. Does not return anything.
<@SQ>	Returns a single quote, for use within quoted attributes.
<@SQL>	Returns last action-generated SQL.
<@STARTROW>	Returns the position of the first row retrieved.
<@STOP>	Immediately ceases execution of the request.
<@SUBSTRING>	Extracts a substring.
<@THROWERROR>	Generates a custom error.
<@TIMER>	Allows you to create and use named elapsed timers.
<@TIMETOSECS>, <@SECSTOTIME>	Converts a time to seconds.
<@TMPFILENAME>	Generates a unique temporary file name.
<@TOGMT>	Transforms a local time to GMT.
<@TOKENIZE>	Sections a string into a one-row array.
<@TOTALROWS>	Returns the total number of rows matched by a Search action.

<@TOUTC>	Transforms a local time to UTC.
<@TRANSPOSE>	Exchanges row and column specifications for values in an array.
<@TRIM>	Returns a string stripped of leading and trailing spaces.
<@TSTOSECS>, <@SECSTOTS>	Converts a timestamp to seconds.
<@UNION>	Returns the union of two arrays.
<@UPPER>	Returns a string converted to uppercase.
<@URL>	Retrieves the specified URL, returns its data, and optionally a variety of additional information.
<@URLDECODE>	Decodes a string encoded in URL format.
<@URLENCODE>	Makes a string compatible for inclusion in a URL.
<@USERREFERENCE>	Returns a value identifying the user executing the application file.
<@USERREFERENCEARGUMENT>	Evaluates to _userReference=<@USERREFERENCE>.
<@USERREFERENCECOOKIE>	Used in default HTTP header of TeraScript when returning results.
<@UUID>	Outputs a Universally Unique Identifier based on RFC 4122 version 4.
<@VAR>	Retrieves the contents of a variable.
<@VARINFO>	Returns information about a variable.
<@VARNAMES>	Returns an array of all variable names for a given scope.
<@VARPARAM>	Explicitly passes a value in the <@CALLMETHOD> Meta Tag.
<@VERSION>	Returns the version number of TeraScript Server.
<@WEBROOT>	Returns the path to the Web server document root.
<@WHILE> </@WHILE>	Provides WHILE loop functionality.
<@XSLT>	Applies and XSLT to a DOM.
<@!>	Allows commenting of application files.

Alphabetical List of Meta Tags, With Attributes

Square brackets [ ] denote optional attributes (or tags, in the case of multi-tag expressions).

<@ABSROW>
<@ACTIONRESULT NAME NUM [ENCODING] [FORMAT]>
<@ADDROWS ARRAY VALUE [POSITION] [SCOPE]>
<@APPFILE [ENCODING]>
<@APPFILENAME [ENCODING]>
<@APPFILEPATH [ENCODING]>
<@APPKEY [ENCODING]>
<@APPNAME [ENCODING]>
<@APPPATH [ENCODING]>
<@ARG NAME [TYPE] [ENCODING] [FORMAT]>
<@ARGNAMES>
<@ARRAY [ROWS] [COLS] [VALUE] [CDELIM] [RDELIM]>
<@ASCII [CHAR]>
<@ASSIGN NAME VALUE [SCOPE] [EXPIRES] [PATH] [DOMAIN] [SECURE]>
<@BIND NAME [DATATYPE] [SCOPE] [BINDTYPE] [PRECISION] [SCALE] [BINDNAME]>
<@BREAK>
<@CALC EXPR [PRECISION] [ENCODING] [FORMAT]>
<@CALLMETHOD OBJECT METHOD [SCOPE] [METHODTYPE] [PARAMTYPES]>
<@CGI [ENCODING]>
<@CHAR CODE [ENCODING]>
<@CHOICELIST NAME TYPE OPTIONS [SIZE] [MULTIPLE] [CLASS] [STYLE] [onBlur] [onClick] [onFocus] [VALUES] [SELECTED] [SELECTEXTRAS] [OPTIONEXTRAS] [TABLEEXTRAS] [TREXTRAS] [TDEXTRAS] [LABELPREFIX] [LABELSUFFIX] [COLUMNS] [ROWS] [ORDER] [ENCODING]>
<@CIPHER ACTION TYPE STR [KEY] [ENCODING][KEYTYPE]>
<@CLASSFILE [ENCODING]>
<@CLASSFILEPATH [ENCODING]>
<@CLEARERRORS>
<@COL [NUM] [ENCODING] [FORMAT]>
<@COLS></@COLS>
<@COLUMN NAME [ENCODING] [FORMAT]>
<@COMMENT></@COMMENT>
<@CONFIGPATH>
<@CONNECTIONS [DSN] [TYPE] [ENCODING] [{array attributes}]>
<@CONTINUE>
<@CREATEOBJECT TYPE OBJECTID [SCOPE] [EXPIRYURL] [INITSTRING] [SYSTEMOBJECT]>
<@CRLF>
<@CURCOL>
<@CURRENTACTION [ENCODING]>
<@CURRENTDATE [ENCODING] [FORMAT]>
<@CURRENTTIME [ENCODING] [FORMAT]>
<@CURRENTTIMESTAMP [ENCODING] [FORMAT]>
<@CURROW>
<@CUSTOMTAGS [SCOPE] [{array attributes}]>
<@DATASOURCESTATUS [DSN] [TYPE] [ENCODING] [{array attributes}]>
<@DATEDIFF DATE1 DATE2 [FORMAT]>
<@DATETOSECS DATE [FORMAT]>
<@DAYS DATE DAYS [ENCODING] [FORMAT]>

<@DBMS [ENCODING]>
<@DEBUG></@DEBUG>
<@DEFINE [NAME] [SCOPE]TYPE [ROWS][COLS]>
<@DELROWS ARRAY [POSITION] [NUM] [SCOPE]>
<@DISTINCT ARRAY [COLS] [SCOPE]>
<@DOCS [FILE] [ENCODING]>
<@DOM VALUE [XPOINTER] [XPATH]>
<@DOMAIN>
<@DOMDELETE OBJECT [SCOPE] [XPOINTER] [XPATH]>
<@DOMINSERT OBJECT [SCOPE] [XPOINTER] [XPATH] [POSITION]><@DOMINSERT>
<@DOMREPLACE OBJECT [SCOPE] [XPOINTER] [XPATH]></@DOMREPLACE>
<@DOMSEARCH OBJECT [SCOPE] [XPOINTER] [XPATH]>
<@DQ>
<@DSDATE DATE [INFORMAT] [ENCODING]>
<@DSTIME TIME [INFORMAT] [ENCODING]>
<@DSTIMESTAMP TS [INFORMAT] [ENCODING]>
<@DSNUM NUM [ENCODING]>
<@DSTYPE [ENCODING]>
<@EDITION [ENCODING]>
<@ELEMENTATTRIBUTE OBJECT ATTRIBUTE [SCOPE] [XPOINTER] [XPATH] [TYPE] [{array attributes}]>
<@ELEMENTATTRIBUTES OBJECT [SCOPE] [XPOINTER] [XPATH] [TYPE] [{array attributes}]>
<@ELEMENTNAME OBJECT [SCOPE] [XPOINTER] [XPATH] [TYPE] [{array attributes}]>
<@ELEMENTVALUE OBJECT [SCOPE] [XPOINTER] [XPATH] [TYPE] [{array attributes}]>
<@EMAIL [COMMAND] NAME SCOPE [PARTID] [FIELDNAME] [FIELDVALUE] [TYPE] [DECODEDATA] [MESSAGE]>
<@EMAILSESSION [COMMAND] [SESSIONID] PROTOCOL SERVER [PORT] [USERNAME] [PASSWORD] [MAILBOX] [MODE] [FIELD] [MESSAGEID] [NAME] [SCOPE]>
<@ERROR PART [ENCODING]>
<@ERRORS></@ERRORS>
<@EXCLUDE></@EXCLUDE>
<@EXIT>
<@FILTER ARRAY EXPR [SCOPE]>
<@FOR [START] [STOP] [STEP] [PUSH]></@FOR>
<@FORMAT STR [FORMAT] [INFORMAT] [ENCODING]>
<@GETPARAM NAME [TYPE] [ENCODING] [FORMAT] [{array attributes}]>
<@HTTPATTRIBUTE NAME [ENCODING]>
<@HTTPREASONPHRASE>
<@HTTPSTATUSCODE>
<@IF EXPR [TRUE] [FALSE]>
<@IF EXPR> [<@ELSEIF EXPR>][<@ELESIFEMPTY VALUE>][<@ELESIFNOTEMPTY VALUE>][<@ELSEIFEQUAL VALUE1VALUE2>][<@ELSE>]</@IF>
<@IFEMPTY VALUE> [<@ELSE>]</@IF>
<@IFEQUAL VALUE1 VALUE2>[<@ELSE>]</@IF>
<@IFNOTEMPTY VALUE> [<@ELSE>]</@IF>
<@INCLUDE FILE>
<@INTERSECT ARRAY1 ARRAY2 [COLS] [SCOPE1] [SCOPE2]>
<@ISALPHA STR>
<@ISALPHANUM STR>
<@ISDATE VALUE>
<@ISDECIMAL STR>

<@ISINT STR>
<@ISMETASTACKTRACE>
<@ISNULLOBJECT OBJECT [SCOPE]>
<@ISNUM VALUE>
<@ISTIME VALUE>
<@ISTIMESTAMP VALUE>
<@KEEP STR CHARS [ENCODING]>
<@LDAPADD SERVER [PORT] [PROTOCOL] [USERNAME] [PASSWORD] DN FILTER ATTRIBUTES [ATTRDELIM] [VALUEDELIM] [TIMEOUT] >
<@LDAPDELETE SERVER [PORT] [PROTOCOL] [USERNAME] [PASSWORD] DN [TIMEOUT] >
<@LDAPMODIFY SERVER [PORT] [PROTOCOL] [USERNAME] [PASSWORD] DN FILTER ATTRIBUTES [ATTRDELIM] [VALUEDELIM] [TIMEOUT] >
<@LDAPSEARCH SERVER [PORT] [PROTOCOL] [USERNAME] [PASSWORD] DN FILTER [FILTERSCOPE] ATTRIBUTES [ATTRDELIM] [VALUEDELIM] [TIMEOUT] [RETURNTYPE]>
<@LEFT STR NUMCHARS [ENCODING]>
<@LENGTH STR>
<@LITERAL VALUE [ENCODING]>
<@LOCATE STR FINDSTR>
<@LOGMESSAGE MESSAGE [LOGLEVEL] [TYPE={ACTIVITY*|EVENT}]>
<@LOWER STR [ENCODING]>
<@LTRIM STR [ENCODING]>
<@MAKEPATH [PATH1] [PATH2] [TYPE]>
<@MAP NAME [SCOPE] VALUE [ENCODING]>
<@MAXROWS>
<@METAOBJECTHANDLERS [{array attributes}]>
<@METASTACKTRACE>
<@MIMEBOUNDARY LEVELID [BOUNDARY]>
<@NEXTVAL NAME [SCOPE] [STEP]>
<@NULLOBJECT>
<@NUMAFFECTED>
<@NUMCOLS [ARRAY]>
<@NUMOBJECTS OBJECT [SCOPE]>
<@NUMROWS [ARRAY]>
<@OBJECTAT OBJECT NUM [SCOPE]>
<@OBJECTS OBJECT ITEMVAR [SCOPE] [ITEMSCOPE] [START] [STOP]>
</@OBJECTS>
<@OMIT STR CHARS [ENCODING]>
<@PAD STR CHAR NUMCHARS [POSITION] [ENCODING]>
<@PLATFORM [ENCODING]>
<@POSTARG NAME [TYPE] [ENCODING] [FORMAT]>
<@POSTARGNAMES>
<@PRODUCT [ENCODING]>
<@PURGE [NAME] [SCOPE]>
<@PURGECACHE [PATH] [TYPES][DOMAIN]>
<@PURGEDEBUG>
<@PURGERESULTS>
<@RANDOM [HIGH] [LOW]>
<@REGEX EXPR STR TYPE>
<@RELOADCONFIG>
<@RELOADCUSTOMTAGS [SCOPE]>
<@REPLACE STR FINDSTR REPLACESTR [POSITION] [ENCODING] [TYPE]>
<@RESULTS [ENCODING]>
<@RIGHT STR NUMCHARS [ENCODING]>
<@ROWS [ARRAY] [SCOPE] [PUSH] [START] [STOP] [STEP]></@ROWS>

<@RTRIM STR [ENCODING]>
<@SCRIPT EXPR [SCOPE]>
<@SCRIPT [SCOPE]></@SCRIPT>
<@SEARCHARG NAME [TYPE] [ENCODING] [FORMAT]>
<@SEARCHARGNAMES>
<@SECSTODATE SECS [ENCODING] [FORMAT]>
<@SECSTOTIME SECS [ENCODING] [FORMAT]>
<@SECSTOTS SECS [ENCODING] [FORMAT]>
<@SERVERNAME>
<@SERVERSTATUS [VALUE] [ENCODING]>
<@SETCOOKIES>
<@SETPARAM NAME VALUE>
<@SLEEP VALUE>
<@SORT ARRAY [COLS] [SCOPE]>
<@SQ>
<@SQL [ENCODING]>
<@STARTROW>
<@STOP>
<@SUBSTRING STR START NUMCHARS [ENCODING]>
<@THROWERROR [NUM] [DESCRIPTION]>
<@TIMER [NAME] [VALUE]>
<@TIMETOSECS TIME [FORMAT]>
<@TMPFILENAME [ENCODING]>
<@TOGMT TS [ENCODING] [FORMAT]>
<@TOKENIZE STR CHARS [CDELIM][RDELIM]>
<@TOTALROWS>
<@TOUTC TS [ENCODING] [FORMAT]>
<@TRANSPOSE ARRAY [SCOPE]>
<@TRIM STR [ENCODING]>
<@TSTOSECS TS [FORMAT]>
<@UNION ARRAY1 ARRAY2 [COLS] [SCOPE1] [SCOPE2]>
<@UPPER STR [ENCODING]>
<@URL LOCATION [BASE] [USERAGENT] [FROM] [ENCODING] [USERNAME] [PASSWORD] [POSTARGS] [POSTARGARRAY] [WAITFORRESULT] [DETAILEDRESPONSE]>
<@URLDECODE STR>
<@URLENCODE STR>
<@USERREFERENCE>
<@USERREFERENCEARGUMENT>
<@USERREFERENCECOOKIE>
<@UUID>
<@VAR NAME [XPOINTER] [XPATH] [SCOPE] [TYPE] [ENCODING] [FORMAT] [{array attributes}]>
<@VARINFO NAME ATTRIBUTE [SCOPE]>
<@VARNAMES SCOPE>
<@VARPARAM NAME [DATATYPE] [SCOPE]>
<@VERSION [ENCODING]>
<@WEBROOT>
<@WHILE [EXPR] [PUSH]></@WHILE>
<@XSLT OBJECT [SCOPE] STYLESHEET= [ENCODING]>
<@! COMMENT>

Meta Tags List by Function

Action/Application File Information

<@ACTIONRESULT>
<@APPFILENAME>
<@CURRENTACTION>
<@DOCS>
<@LOGMESSAGE>
<@RESULTS>
<@SQL>

Application Scope

<@APPKEY>
<@APPNAME>
<@APPPATH>

Array Operations

<@ADDROWS>
<@ARRAY>
<@ASSIGN>
<@DELROWS>
<@DEFINE>
<@DISTINCT>
<@FILTER>
<@INTERSECT>
<@MAP>
<@NUMCOLS>
<@REGEX>
<@ROWS> </@ROWS>
<@SORT>
<@TOKENIZE>
<@TRANSPOSE>
<@UNION>
<@VAR>
<@VARINFO>

Conditionals

<@IF>, <@ELSEIF>, <@ELSEIFEMPTY>, <@ELSEIFNOTEMPTY>,<@ELSEIFEQUAL>,
</@IF>
<@IFNOTEMPTY> <@ELSE> </@IF>
@ELSEIFEMPTY
<@IFEMPTY> <@ELSE> </@IF>@ELSEIFEMPTY
<@IFEQUAL> <@ELSE> </@IF>@IF

Custom Meta Tags

<@RELOADCUSTOMTAGS>
<@CUSTOMTAGS>

Data Sources

<@BIND>
<@CONNECTIONS>
<@DATASOURCESTATUS>
<@DBMS>
<@DSDATE>, <@DSTIME>, <@DSTIMESTAMP>
<@DSNUM>
<@DSTYPE>
<@SQL>

Database Output

<@ABSROW>
<@COL>
<@COLS> </@COLS>
<@COLUMN>
<@CURCOL>
<@CURROW>
<@FORMAT>
<@MAXROWS>
<@NUMAFFECTED>
<@NUMROWS>
<@PURGEDEBUG>
<@PURGERESULTS>
<@ROWS> </@ROWS>
<@STARTROW>
<@TOTALROWS>

Date and Time

<@CURRENTDATE>, <@CURRENTTIME>, <@CURRENTTIMESTAMP>
<@DATEDIFF>
<@DATETOSECS>, <@SECSTODATE>
<@DAYS>
<@DSDATE>, <@DSTIME>, <@DSTIMESTAMP>
<@ISDATE>, <@ISTIME>, <@ISTIMESTAMP>
<@SECSTODATE>, <@SECSTOTIME>, <@SECSTOTS>
<@TIMER>
<@TIMETOSECS>, <@SECSTOTIME>
<@TOGMT>
<@TOUTC>
<@TSTOSECS>, <@SECSTOTS>

Document Instance (XML)

<@ASSIGN>

<@DEFINE>
<@DOCS>
<@DOM>
<@DOMDELETE>
<@DOMINSERT>
<@DOMREPLACE>
<@DOMSEARCH>
<@ELEMENTATTRIBUTE>
<@ELEMENTATTRIBUTES>
<@ELEMENTNAME>
<@ELEMENTVALUE>
<@VAR>

Email

<@DEFINE>
<@EMAIL>
<@EMAILSESSION>
<@MIMEBOUNDARY>

Error Handling

<@CLEARERRORS>
<@ERROR>
<@ERRORS> </@ERRORS>
<@ISMETASTACKTRACE>
<@METASTACKTRACE>
<@THROWERROR>

File Access

<@APPFILE>
<@APPFILENAME>
<@APPFILEPATH>
<@CLASSFILE>
<@CLASSFILEPATH>
<@INCLUDE>
<@TMPFILENAME>
<@WEBROOT>

Formatting

<@FORMAT>
<@XSLT>

HTML Processing

<@CHOICELIST>
<@COMMENT> </@COMMENT>
<@DEBUG> </@DEBUG>
<@EXCLUDE> </@EXCLUDE>

<@EXIT>
<@!>

HTTP Processing

<@HTTPATTRIBUTE>
<@HTTPSTATUSCODE>

LDAP

<@LDAPADD>
<@LDAPDELETE>
<@LDAPMODIFY>
<@LDAPSEARCH>

Loop Processing

<@BREAK>
<@CONTINUE>
<@FOR> </@FOR>
<@ROWS> </@ROWS>
<@OBJECTS></@OBJECTS>
<@WHILE> </@WHILE>

Numeric Operations

<@CALC>
<@DSNUM>
<@ISNUM>
<@NEXTVAL>
<@RANDOM>

Objects

<@CALLMETHOD>
<@CREATEOBJECT>
<@ISNULLOBJECT>
<@METAOBJECTHANDLERS>
<@NULLOBJECT>
<@NUMOBJECTS>
<@OBJECTAT>
<@OBJECTS></@OBJECTS>
<@VARPARAM>

Paths

<@APPFILE>
<@APPFILEPATH>
<@CGI>
<@CLASSFILE>

<@CLASSFILEPATH>
<@CONFIGPATH>
<@MAKEPATH>

Server

<@SERVERNAME>
<@SERVERSTATUS>
<@SLEEP>

Script Execution

<@SCRIPT>

String Operations

<@ASCII>
<@CHAR>
<@CIPHER>
<@DQ>, <@SQ>
<@ISALPHA>
<@KEEP>
<@LEFT>
<@LENGTH>
<@LOCATE>
<@LOWER>
<@LTRIM>
<@OMIT>
<@PAD>
<@REGEX>
<@REPLACE>
<@RIGHT>
<@RTRIM>
<@SUBSTRING>
<@TOKENIZE>
<@TRIM>
<@UPPER>
<@URLENCODE>

TeraScript Class Files

<@CLASSFILE>
<@CLASSFILEPATH>
<@GETPARAM>
<@SETPARAM>

TeraScript Information

<@PLATFORM>
<@PRODUCT>
<@VERSION>

CHAPTER 11

Using DLLs With TeraScript

11

Programmer Reference for Extending the Functionality of TeraScript Using DLLs

This Chapter provides information on creating Dynamic Link Libraries (DLLs) for use with the External action when executing TeraScript application files on the Windows platform. This information is provided for those programmers who want to extend the functionality of TeraScript Server through the use of DLLs.

TExtParamBlock

TeraScript passes each function the following parameter block:

```
struct TExtParamBlock{
       DWORD ThreadId;
       DWORD CurrRow;
       DWORD CurrColumn;
       VOID *UserData;
       VOID *Reserved;
};
```

TeraScript uses this "extension parameter block" to communicate the current state to the DLL. The DLL uses it to track user data between invocations of the DLL. The members of the parameter block are:

- `DWORD ThreadId;`

  The ID of the calling thread allocated by TeraScript. The value is set by TeraScript; it may not be changed by the DLL.

- `DWORD CurrRow;`

  The row number currently processed by TeraScript. This value is incremented by TeraScript as it iterates through each row of the result set. Starting value is 0.

- `DWORD CurrColumn;`

  The column number currently processed by TeraScript. This value is incremented by TeraScript as it iterates through each column of a particular row of the result set. Starting value is 0.

- `VOID *UserData;`

  Contains any user defined data. If the DLL requires some memory on a per query basis, use the UserData member to keep a reference to the memory block. UserData is usually assigned at the time of `ExtSrcConnect` call. It must be freed in the `ExtSrcDisconnect` function.

- `VOID *Reserved;`

  Reserved for future use by TeraScript. Do not reference or set this member.

DLL Functions

DLL writers must implement five functions in their DLL. A sixth function, used to process errors, is optional. TeraScript calls these functions to process specific events.

The prototypes for these functions are defined in the `ExtSrc.h` file, included with TeraScript.

These DLL functions are:

- `extern "C" _export int ExtSrcConnect(TExtParamBlock *param_block);`

  This function is called when the external data source is connected, usually the first time the External action that references the DLL is executed. The `UserData` member of the `TExtParamBlock` structure should be initialized at this point.

  This function must return one of the following values:

  `EXT_SRC_SUCCESS` (if connection is successful)

  `EXT_SRC_ERROR` (otherwise)

- `extern "C" _export int ExtSrcDisconnect(TExtParamBlock *param_block);`

  This function is called when the external data source is disconnected, usually when the TeraScript Service is stopped. This function provides the last opportunity to deallocate any memory referenced by the `UserData` member of the parameter block.

  This function must return one of the following values:

  `EXT_SRC_SUCCESS` (if disconnection is successful)

  `EXT_SRC_ERROR` (otherwise)

- `extern "C" _export int ExtSrcExecuteQuery(TExtParamBlock *param_block, char *p1, char *p2, char *p3);`

  The `ExtSrcExecuteQuery` function is called once for each time the External action is executed by TeraScript Server. This function either returns an error code or the number of columns in the result set arising from the execution of the DLL. The number of columns is used by the TeraScript Server to control when the `ExtSrcGetNextColumn` function is called.

The declaration of this function depends on the number of parameters you intend to pass to the DLL. After the `param_block` parameter you need to include a char * parameter for each parameter defined in the External action window in TeraScript Editor. For example, the prototype shown above is for a DLL that has three parameters defined for it in the External action.

This function must return one of the following values:

`EXT_SRC_ERROR` (in case of error)

result set quantity (zero or greater number identifying the number of columns in the result set)

- `extern "C" _export int ExtSrcFetchNextRow(TExtParamBlock *param_block);`

  This function is called by TeraScript Server once for each row of the result set created by the `ExtSrcExecuteQuery` function. The result of this function determines the number of times it is called: TeraScript Server continues to call `ExtSrcFetchNextRow` until the function returns `EXT_SRC_NODATA` or `EXT_SRC_ERROR`.

  This function does not return data to TeraScript Server. It should be used by the DLL to load or prepare the data for retrieval. After calling this function, the `ExtSrcGetNextColumn` function is called to retrieve the data from each column. `ExtSrcGetNextColumn` is called once for each column in the result set; the number of columns is determined by the result of the `ExtSrcExecuteQuery` function.

  This function must return one of the following values:

  `EXT_SRC_SUCCESS` (if the row is retrieved successfully)

  `EXT_SRC_NODATA` (if there are no rows remaining to return)

  `EXT_SRC_ERROR` (if an error occurs)

- `extern "C" _export int ExtSrcGetNextColumn(TExtParamBlock *param_block, UCHAR *buffer, DWORD blen, DWORD *actlen);`

  This function is called by TeraScript Server once for each column of the result set for each row fetched by the `ExtSrcFetchNextRow` function.The number of columns is determined by the result of the `ExtSrcExecuteQuery` function. If the value of the `CurrColumn` member of `param_block` is equal or greater than the value returned by `ExtSrcExecuteQuery`, `ExtSrcGetNextColumn` returns `EXT_SRC_ERROR`.

This function has the following parameters:

`TExtParamBlock *param_block` (pointer to the external action parameter block)

`UCHAR *buffer` (pointer to a 32K buffer allocated by TeraScript)

`DWORD blen` (size of the buffer allocated by TeraScript (currently set to 32K))

`DWORD *actlen` (actual size of the data written by the DLL to the buffer plus one)

This function must return one of the following values:

`EXT_SRC_SUCCESS` (if the column's data is retrieved successfully)

`EXT_SRC_ERROR` (if an error occurs)

- `extern "C" _export int ExtSrcErrorCode(TExtParamBlock *param_block);`

  This is an optional function. If implemented, TeraScript calls `ExtSrcErrorCode` whenever one of the other DLL functions returns `EXT_SRC_ERROR` to TeraScript.