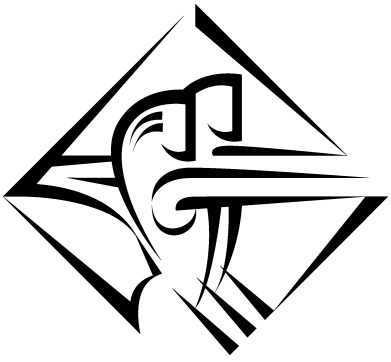
Tango 3



Meta Tags and Configuration Variables

January, 1999

Pervasive Software
12365 Riata Trace Parkway, Building II
Austin, Texas 78727 USA

Telephone: (512) 231-6000
Toll Free: 1 800 287-4383
Fax: (512) 231-6010

Email: info@pervasive.com
Web: http://www.pervasive.com/

PERVASIVE SOFTWARE

Pervasive Software Inc.

Tango(TM) License Agreement

IMPORTANT: DO NOT INSTALL THE ENCLOSED SOFTWARE UNTIL YOU HAVE READ THIS LICENSE AGREEMENT. BY INSTALLING THE SOFTWARE, OR AUTHORIZING ANY OTHER PERSON TO DO SO, YOU, AND SUCH OTHER PERSON, IF APPLICABLE ACCEPT THE TERMS AND CONDITIONS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO THE TERMS OF THIS LICENSE AGREEMENT, RETURN THE ENTIRE PACKAGE WITHIN TEN DAYS OF RECEIPT.

1 DEFINITIONS.

a. "Pervasive" means Pervasive Software Inc., a Delaware corporation, 12365 Riata Trace Parkway, Building II, Austin, Texas (U.S.A.) 78727.

b. "You" or "Your Company" means the person or business entity which is licensing the Software pursuant to this Agreement.

c. "Software" means all of the software You or Your Company have received from Pervasive with this License.

d. "Documentation" means the manuals and any other printed material provided by Pervasive with the Software.

e. "License" means the license purchased and granted pursuant to this Agreement.

f. "Effective Date" means the date on which the Software was licensed by You or Your Company.

2 LICENSE AND PROTECTION.

2.1 License Grant. Pervasive grants to You or Your Company, subject to the following terms and conditions, a limited nonexclusive, nontransferable, revocable right to use the Software solely for Your internal development and Your internal testing purposes only on a single-user, desk-top device. You are prohibited from deploying any application programs to other parties that You develop using the Software.

2.2 Documentation. Documentation may not be copied. Pervasive reserves all rights not expressly granted to You or Your Company.

2.3 Protection of Software. You and Your Company agree to take all reasonable steps to protect the Software and Documentation from unauthorized copying or use. The Software source code represents and embodies trade secrets of Pervasive and/or its licensors. The source code and embodied trade secrets are not licensed to You or Your Company and any modification or addition thereto, or deletion therefrom is strictly prohibited. You and Your Company agree not to disassemble, decompile, or otherwise reverse engineer the Software in order to discover the source code and/or the trade secrets contained in the source code.

3 COPIES. You may make a single copy of the Software for archival purposes only. All proprietary rights notices must be faithfully reproduced and included on all copies. You may not copy the Documentation.

4 OWNERSHIP. Ownership of, and title to, the Software and Documentation (including any copies) shall be vested solely in Pervasive. Copies are provided to You or Your Company only to allow You or Your Company to exercise Your or Your Company's rights under this Agreement. Only the License is purchased by You or Your Company, as applicable.

5 RESTRICTIONS. Except as expressly authorized in this Agreement, You and Your Company agree not to use, rent, lease, sublicese, distribute, transfer, copy, reproduce, display, modify, create derivative works of, time share or dispose of the Software or Documentation or any part thereof. You or Your Company, as applicable, may use the Software and Documentation solely for your internal business purposes.

6 ASSIGNMENT. You or Your Companies may not assign or otherwise transfer in whole or in part or in any manner any rights, obligations, or any interest in or under this Agreement without Pervasive Software's prior written consent and any attempted assignment will be void. A merger or other acquisition by a third party will be treated as an assignment.Pervasive may at any time and without Your or Your Company's consent assign all or a portion of its rights and duties under this Agreement to a company or companies wholly owning, owned by, or in common ownership with Pervasive.

7 TERM, TERMINATION. This Agreement is effective from the date You or Your Company open the software envelope and will remain in force until terminated. You may terminate this License at any time by destroying the Documentation and the Software together with all copies and adaptations. This Agreement shall also automatically terminate if You or Your Company breach any of the terms or conditions of this Agreement. You and Your Company agree to destroy the original and all copies of the Software and Documentation, or to return them to Pervasive upon termination of this License. Sections 2.2, 4 and 5 of this Agreement shall survive any termination hereof.

8 LIMITED WARRANTY AND LIMITED LIABILITY.

8.1 Magnetic Media and Documentation. Pervasive warrants that if the magnetic media or Documentation are in a damaged or physically defective condition at the time that the License is purchased and if they are returned to Pervasive within 90 days of purchase, then Pervasive will provide You or Your Company with replacements at no charge.

8.2 Disclaimer of Warranty. PERVASIVE LICENSES THE SOFTWARE PRODUCT TO YOU OR YOUR COMPANY UNDER THIS AGREEMENT SOLELY ON AN "AS IS" BASIS. PERVASIVE MAKES NO OTHER REPRESENTATIONS, CONDITIONS OR WARRANTIES WHATSOEVER, EITHER EXPRESS OR IMPLIED, REGARDING THE SOFTWARE; PERVASIVE EXPRESSLY STATES AND YOU OR YOUR COMPANY ACKNOWLEDGES THAT PERVASIVE DOES NOT MAKE ANY REPRESENTATIONS, CONDITIONS OR WARRANTIES, INCLUDING, FOR EXAMPLE, WITH RESPECT TO MERCHANTABILITY, TITLE, OR FITNESS FOR ANY PARTICULAR PURPOSE OR ARISING FROM COURSE OF DEALING OR USAGE OF TRADE, AMONG OTHERS.

9 GENERAL CONDITIONS.

9.1 Governing Law. This License Agreement will be governed by, and interpreted in accordance with, the laws of the State of Texas (U.S.A.) exclusive of its choice of law provisions. This Agreement expressly "excludes the United Nations Convention on Contracts for the International Sale of Goods."

9.2 Complete Understanding. This License Agreement sets forth the entire understanding and agreement between You or Your Company and Pervasive and may be amended only in writing signed by both parties. NO VENDOR, DISTRIBUTOR, DEALER, RETAILER, SALES PERSON OR OTHER PERSON IS AUTHORIZED TO MODIFY THIS AGREEMENT OR TO MAKE ANY WARRANTY, REPRESENTATION OR PROMISE WHICH IS DIFFERENT THAN, OR IN ADDITION TO, THIS AGREEMENT ABOUT THE SOFTWARE.

9.3 Waiver. No waiver of any right under this Agreement shall be effective unless in writing, signed by a duly authorized representative of Pervasive. No waiver of any past or present right arising from any breach or failure to perform shall be deemed to be a waiver of any future right arising under this Agreement.

9.4 Severability. If any provision in this Agreement is held invalid or unenforceable, that provision shall be construed, limited, modified or, if necessary, severed, to the extent necessary, to eliminate its invalidity or unenforceablility, and the other provisions of this Agreement shall remain unaffected.

9.5 Export Controls. None of the Software or underlying information or technology may be downloaded or otherwise exported or reexported (i) into (or to a national or resident of) Cuba, Iraq, Libya, North Korea, Iran, or any other country to which the U.S. has embargoed goods; or (ii) to any person or entity on the U.S. Treasury Department's list of Specially Designated Nationals, or the U.S. Commerce Department's Table of Denial Orders, or the

U.S. Commerce Department's Entity List of Missile, Nuclear, and Chemical and Biological Weapons Proliferators, or the U.S. Department of State's Foreign Terrorist Organization List. You agree to the foregoing and you represent the warrant that you are not located in, under the control of, or a national or resident of such country or on any such list. The Software may also be subject to U.S. laws and export regulations of the U.S. government that require an explicit export license prior to any export or reexport of the Software. You agree to obtain any such explicit export license that may be required.

9.6 U.S Government End Users. The Software is a "commercial item," as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of "commercial computer software" and "commercial computer software documentation," as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire the Software with only those rights set forth herein. Contractor / Manufacturer is Pervasive Software Inc., 12365 Riata Trace Parkway, Building II, Austin, Texas (U.S.A.) 78727.

9.7 Consequential Damages. WITHOUT LIMITING THE FOREGOING,IN NO EVENT WILL PERVASIVE BE LIABLE TO YOU OR YOUR COMPANY FOR INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO THIS LICENSE AGREEMENT, WHETHER UNDER THEORY OF WARRANTY, TORT, PRODUCTS LIABILITY OR OTHERWISE.

9.8 High Risk Activities. The Software is not fault-tolerant and is not designed, manufactured or intended for use or resale as on-line control equipment in hazardous environments regarding fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems, in which the failure of the Software could lead directly to death, personal injury, or severe physical or environmental damage ("High Risk Activities"). Pervasive and its suppliers specifically disclaim any express or implied warranty of fitness for High Risk Activities.

9.9 English will be the controlling language of this Agreement.

Microsoft Software Disclaimer

NO OTHER WARRANTIES, TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, MICROSOFT AND ITS SUPPLIERS DISCLAIM ALL OTHER WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WITH REGARD TO THE SOFTWARE PRODUCT, AND ANY ACCOMPANYING HARDWARE. THIS LIMITED WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY HAVE OTHERS, WHICH VARY FROM STATE / JURISDICTION TO STATE / JURISDICITION.

NO LIABILITY FOR CONSEQUENTIAL DAMAGES, TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL MICROSOFT OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WIHTOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE PRODUCT, EVEN IF MICROSOFT HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME STATES AND JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

Apple Software Disclaimer

APPLE COMPUTER, INC. ("APPLE") MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATIONS THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A SPECIFIC PURPOSE, REGARDING THE APPLE SOFTWARE. APPLE DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE APPLE SOFTWARE IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE APPLE SOFTWARE IS ASSUMED BY YOU. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME STATES. THE ABOVE EXLUSION MAY NOT APPLY TO YOU.

IN NO EVENT WILL APPLE, ITS DIRECTORS, OFFICERS, EMPLOYEES OR AGENTS BE LIABLE TO YOU FOR ANY CONSEQUENTIAL, INCIDENTAL OR INDIRECT DAMAGES (INCLUDING DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTIONS, LOSS OF BUSINESS INFORMATION, AND THE LIKE) ARISING OUT OF THE USE OR INABILITY TO USE THE APPLE SOFTWARE EVEN IF APPLE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMJAGES. BECAUSE SOME STATES DO NOT ALLOW THE EXLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU. Apple's liability to you for actual damages from any cause whatsoever, and regardless of the form of the action (whether in contract, tort [including negligence], product liability or otherwise), will be limited to $50.

Copyright Notices

The Tango manual, program design, and design concepts are copyrighted, with all rights being subject to the limitations and restrictions imposed by the copyright laws of the United States of America and Canada. Under the copyright laws, this manual may not be copied, in whole or part, including translation to another language or format, without the express written consent of Pervasive Software Inc.

Copyright © 1999 Pervasive Software Inc. All rights reserved.

Tango Enterprise, the Tango software, Tango Editor, Tango Server, Tango CGI, the documentation, and associated materials are © 1992–1999 Pervasive Software Inc. All rights reserved. Tango is a trademark of Pervasive Software Inc.

FileMaker is a registered trademark and the FileMaker Pro design is a trademark of FileMaker Corporation.

Microsoft, Windows, Windows NT, Windows 95, and the Windows logo are registered trademarks or trademarks of the Microsoft Corporation in the United States and / or other countries.

All other trademarks mentioned are the property of their respective owners.

JavaScript 1.2 Compatible. Portions © Netscape Communications Corporation 1996, All Rights Reserved.

THE LICENSED ELEMENTS ARE LICENSED "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, PERFORMANCE, MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE OR NON-INFRINGEMENT. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY LICENSEE. SHOULD THE SOFTWARE PROVE DEFECTIVE, LICENSEE ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY SHALL NETSCAPE OR ITS SUPPLIERS BE LIABLE TO LICENSEE OR ANY OTHER PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING WITHOUT LIMITATION ANY COMMERCIAL DAMAGES OR LOSSES, EVEN IF NETSCAPE HAS BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES.

Calendar Conversion Code. Copyright 1993–1995, Scott E. Lee, all rights reserved.
Permission granted to use, copy, modify, distribute and sell so long as the above copyright and this permission statement are retained in all copies. THERE IS NO WARRANTY – USE AT YOUR OWN RISK.

Regex. Copyright © 1992 Henry Spencer.
Copyright © 1992, 1993
The Regents of the University of California. All rights reserved.
This code is derived from software contributed to Berkeley by Henry Spencer of the University of Toronto.
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and / or other materials provided with the distribution.

3. All advertising materials mentioning features or use of this software must display the following acknowledgment:
This product includes software developed by the University of California, Berkeley and its contributors.

4. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Portions of this software are copyrighted by INTERSOLV, Inc., 1991–1996.

Table of Contents

Introduction

An Overview of This Manual

This manual gives detailed explanations of all meta tags (Chapter 1, which begins on page 3) and configuration variables (Chapter 2, which begins on page 193) that are used to construct application files and configure Tango Server. It is intended as a reference for the user who is familiar with Tango.

Some topics in this manual apply only to Tango Enterprise for Macintosh, Tango Enterprise for Windows, Tango Server on UNIX, or Tango for FileMaker (Mac OS).

The Mac™ OS, Microsoft™ Windows™, UNIX™, and FileMaker Pro™ graphics identify those topics, respectively; otherwise, topics apply equally to all versions.

For details of how to install Tango, see the *Getting Started Guide*.

For details of how to use Tango, see the *User's Guide*.

The uses of meta tags and configuration variables within Tango are discussed in the *User's Guide*. See the following chapters:

- "Using Meta Tags"
- "Working With Variables," especially the section "Using Configuration Variables"
- "Using Tango Server," especially the section "Configuring Tango Server," which discusses configuration variables.

CHAPTER 1

Meta Tags

1

A Reference to Tango Server Meta Tags

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

Meta tags are resolved by Tango Server when your application file is executed. For more information on using meta tags, see Chapter 6, “Using Meta Tags” in the *User’s Guide*.

This chapter covers the following topics:

- what’s new in Tango meta tags
- basic meta tag syntax
- the `ENCODING` attribute
- the `FORMAT` attribute
- terminology changes and additions
- listing of meta tags by function
- where you can use meta tags
- alphabetical table of meta tags and meta tags with attributes
- alphabetical reference to all meta tags, their function, syntax and explanation.

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
  - **Maximum Matches** and **Start Match** fields in Search and Direct DBMS actions
  - External action parameters
  - File action parameters
  - Assign actions (both name and value)
  - If action parameters
  - custom column references used in database actions
  - SQL entered into the Direct DBMS action window
- HTML 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**.

What's New in Tango Meta Tags

There are many new meta tags in this version of Tango. They fit into all categories of functionality: URL, form and CGI values, numeric operations, conditionals, database output, data source information, date and time, user variables, string operations, and array. As well, increased functionality has been given to meta tags that existed in previous versions of Tango. The following are a few of the upgrades and new inclusions to Tango meta tags.

- `<@CALC>` was a simple addition, subtraction, division and multiplication calculator in previous versions of Tango. In Tango 3.0, `<@CALC>` has a new breadth of functionality: it can calculate and resolve complex comparisons, mathematical operations, and logical processes.
- `<@COLUMN>` is now recognized by Tango Server, whereas in previous versions it could only be used in Tango Editor and was converted to `<@COL>` automatically. This change means that you may now use `<@COLUMN>` tags in files included with `<@INCLUDE>`.
- `<@FORMAT>` and the `FORMAT` attribute permit you to control how values returned by Tango are formatted.
- The `<@ARRAY>` tag lets you create two-dimensional arrays for storage in variables.
- There are many new meta tags for working with text strings, including tags for extracting substrings, and finding and replacing text in a value.
- Date and time operations are now easier with the addition of tags for adding and subtracting days to a date, converting date and time values to seconds and back, and testing the validity of date and time string values.

Terminology Changes and Additions

There have been some terminology changes within Tango. Because a query document is now referred to as an *application file*, the meta tags <@QUERYDOC> and <@QUERYPATH> have become <@APPFILE> and <@APPFILEPATH>. There has been no change to what these tags do.

The meta tag <@GLOBAL> has been changed to <@VAR>. This reflects a terminology change from globals to *variables.*

The term *parameter*, used to describe the name / value pairs that made up the syntax of a meta tag, is now *attribute.* This was changed to establish parallelism with existing name / value pair terminology in HTML.

Format of Meta Tags

Syntax

The basic syntax for Tango meta tags is:

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

- The opening "<" is a characteristic of tag languages, including HTML. The "@" symbol distinguishes Tango meta tags.

  At least one space must occur between the 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 tags anywhere a space occurs. For example,

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

  is valid Tango meta tag syntax.

- There is no space allowed before or after the equals (=) sign. 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 Tango meta tag syntax:

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

Naming Attributes

The current version of Tango uses attribute names in meta tags. This is a departure from previous versions of Tango that recognized attributes by their position in the meta tag and did not use attribute names. For example, when using the `<@POSTARG>` tag, previous versions of Tango evaluated the first attribute as the name of the

post argument to get, and the second as the type of encoding to perform on the value returned. The tag looked like this, `<@POSTARG foo METAHTML>`.

In Tango 3.0, all attributes have names. `<@POSTARG>` is now written as, `<@POSTARG NAME="foo" ENCODING="METAHTML">`, or `<@POSTARG ENCODING="METAHTML" NAME="foo">`. The order of the attributes does not matter if the attributes are named.

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.

Using the `<@POSTARG>` example, `<@POSTARG homer>` is still valid in Tango 3.0, because the `NAME` attribute is required, and its designated position is first. But, 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 Tango, the best method to adopt is to enter all attribute names, for example, `<@POSTARG NAME="homer" ENCODING="NONE">`.

Attribute Value Length

The length of each meta tag attribute value is limited only by available memory. Previous versions of Tango had an attribute value limit of 255 characters.

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, Tango would incorrectly interpret `Zip` as the attribute name and `Code` as the start of another attribute. Tango 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 Tango that there is no value).

Note Quotes are not necessary when you are using only meta tags as the attribute value. Tango knows that meta tags begin with <@ and end with >, so no quotes are necessary to delimit.

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:

- 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">'>">
  ```

For more information, see "<@DQ>, <@SQ>" on page 88.

- 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 45 and "<@IF>" on page 103.

- The exceptions to the last rule are the expressions specified for <@CALC> and <@IF> 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).

Encoding Attribute

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

If no encoding attribute is specified, values returned by meta tags used in Results HTML, No Results HTML, and Error HTML undergo a process of conversion so they appear correctly in the user's Web browser. For example, < is converted to <.

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 without translation. The main use for this attribute is for displaying HTML stored in database fields and variables.

For example:

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

Note This attribute value was called "HTML" in previous versions of Tango.

METAHTML

The METAHTML attribute value of the ENCODING attribute performs the same function as NONE but also looks for Tango 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 Tango to replace return, line feed, and return/line feed combinations in the value with <BR> tags. Normally, Web browsers ignore line breaks in HTML. Use this attribute to force the display of returns in database values.

MULTILINEHTML

The MULTILINEHTML attribute value lets you combine the functions of NONE and MULTILINE. This formatting attribute is particularly useful for formatting data entered by users who have used HTML tags for character formatting (for example, bolding and italicization), but who have not used <BR> or <P> tags to properly indicate line or paragraph endings.

URL

The URL formatting attribute value tells Tango 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.

For more information, see "Direct DBMS SQL Auto-Encoding" in Chapter 18 of the *User's Guide* and "noSQLEncoding" on page 212.

Tango 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>">

<@ASSIGN NAME=mySQL VALUE="<@ARG cust_name
ENCODING=sql><@SQ>"
```

Format Attribute

A new attribute, `FORMAT`, 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 a comma-delimited list of values in this order:

```
FORMAT=num:   grouping,
              grouping separator,
              fractional digits,
              fraction separator,
              positive prefix,
              positive suffix,
              negative prefix,
              negative suffix
```

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 Tango 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.

The list items are described as follows:

- **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”.

  ```
  3-1-2  1234567,89,0,123
  ```

  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”).
- **grouping separator** defines the character that separates the groupings. In the previous example, it is a comma. It may be multiple characters.
- **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.
- **fraction separator**, may be multiple digits, and it is displayed even if no fractional digits are present. This is similar to a decimal place separator.
- **positive prefix** determines the prefix used if the number is positive (for example, +).
- **positive suffix** determines the suffix used if the number is positive.
- **negative prefix** determines the prefix used if the number is negative (for example, -).
- **negative suffix** determines the prefix used if the number is negative.

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.1,234.56C
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

Format*	Sample Output	Input Restrictions
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, 1998 would be formatted as "1998-12-01").

For more information, see "dateFormat, timeFormat, timestampFormat" on page 201.

Tango 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 Tango—<@CURRENTTIME>, <@CURRENTIMESTAMP>, and <@CURRENTDATE>—can be used with the new FORMAT attribute or with their old formatting.

There is only one datetime-class synonym: datetime:http.

```
datetime:http = datetime:%A, %d-%b-%Y %H:%M:%S GMT.
```

For example, if the string 1998-09-29 12:34:56 were formatted with datetime:http, the output format would be "Monday, 29-Sep-1998 12:34:56 GMT".

Note datetime:http formatting does not make any adjustments to the time value to correct to GMT time. It simply outputs the input timestamp in the HTTP format. To convert a local time to GMT, use <@TOGMT>.

For more information, see "<@TOGMT>" on page 167.

Alphabetical List of Meta Tags

New meta tags are denoted by a grey box, as are old meta tags with new names.

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 name of the current application file.
<@APPFILEPATH>	Returns the path to the current application file, excluding the application file name, but including the trailing slash
<@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.
<@BREAK>	Ends execution of a loop.
<@CALC>	Returns the result of a calculation.
<@CGI>	Returns the full path and name of the Tango CGI.
<@CGIPARAM>	Evaluates to the specified CGI attribute.
<@CGIPATH>	Returns the path to the directory containing the Tango CGI.
<@CHAR>	Returns the character that has the specified ASCII value.
<@CIPHER>	Performs encryption/decryption on strings.
<@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 Tango application files.
<@CONTINUE>	Ends the current iteration of a loop.
<@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 currently 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.
<@DATEDIFF>	Returns the number of days between the two dates specified.
<@DATETOSECS>	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.
<@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.
<@DQ>	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.
<@ERROR>	Returns the value of the named error component of the current error.
<@ERRORS> </@ERRORS>	In conjunction with <@ERROR>, allows iteration 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.
<@IF> <@ELSEIF> <@ELSEIFEMPTY> <@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.
<@INCLUDE>	Returns the contents of a specified file.
<@INTERSECT>	Returns the intersection of two arrays.
<@ISDATE> <@ISTIME> <@ISTIMESTAMP>	Checks whether a value is a valid date, time, or timestamp.
<@ISNUM>	Checks whether a value is a valid number.
<@KEEP>	Returns a string stripped of specified characters.
<@LEFT>	Returns the first *n* characters from a string.
<@LENGTH>	Returns the number of characters in a string.
<@LITERAL>	Causes Tango to suppress meta tag substitution for the supplied value.
<@LOCATE>	Returns the starting position of a substring in a string.

<@LOWER>	Converts a string to lowercase.
<@LTRIM>	Returns string stripped of leading spaces.
<@MAXROWS>	Returns the value specified in the Maximum Matches field of a Search or Direct DBMS action.
<@NEXTVAL>	Increments a variable and returns the value.
<@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.
<@NUMROWS>	Returns the number of rows retrieved by an action or in a specified array.
<@OMIT>	Returns a string stripped of specified characters.
<@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 the server's product type.
<@PURGE>	Removes one or all variables from a scope.
<@PURGERESULTS>	Empties the currently accumulated Results HTML.
<@RANDOM>	Returns a random number.
<@REGEX>	Finds strings using regular expressions.
<@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> </@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>	Converts seconds to a date.
<@SECSTOTIME>	Converts seconds to a time.

<@SECSTOTS>	Converts seconds to a timestamp.
<@SERVERSTATUS>	Returns status information about Tango Server.
<@SETCOOKIES>	Returns the correct Set-Cookie lines to set the values of cookie variables.
<@SORT>	Returns the input array, sorted by the specified column(s).
<@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.
<@SUBSTRING>	Extracts a substring.
<@TIMER>	Allows you to create and use named elapsed timers.
<@TIMETOSECS>	Converts a time to seconds.
<@TMPFILENAME>	Generates a unique temporary file name.
<@TOGMT>	Transforms a local time to GMT.
<@TOKENIZE>	Provides you with a way of sectioning a string into multiple pieces.
<@TOTALROWS>	Returns the total number of rows matched by a Search action.
<@TRANSPOSE>	Exchanges row and column specifications for values in an array.
<@TRIM>	Returns a string stripped of leading and trailing spaces.
<@TSTOSECS>	Converts a timestamp to seconds.
<@UNION>	Returns the union of two arrays.
<@UPPER>	Returns a string converted to uppercase.
<@URL>	Retrieves the specified URL and returns its data.
<@URLENCODE>	Makes a string compatible for inclusion in a URL.
<@USERREFERENCE>	Returns a value identifying the user executing the application file.
<@USERREFERENCE ARGUMENT>	Evaluates to _userReference=<@USERREFERENCE>.
<@USERREFERENCE COOKIE>	Used in default HTTP header of Tango when returning results.
<@VAR>	Retrieves the contents of a variable.
<@VARINFO>	Returns information about a variable.

<@VARNAMES>	Returns an array containing all variable names for a given scope.
<@VERSION>	Returns the version number of Tango Server.
<@!>	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 [FORMAT] [ENCODING]>
<@ADDROWS ARRAY VALUE [POSITION] [SCOPE]>
<@APPFILE [ENCODING]>
<@APPFILENAME [ENCODING]>
<@APPFILEPATH [ENCODING]>
<@ARG NAME [TYPE] [FORMAT] [ENCODING]>
<@ARGNAMES>
<@ARRAY [ROWS] [COLS] [VALUE] [CDELIM] [RDELIM]>
<@ASCII CHAR>
<@ASSIGN NAME VALUE [SCOPE]>
<@BREAK>
<@CALC EXPR [PRECISION] [FORMAT] [ENCODING]>
<@CGI [ENCODING]>
<@CGIPARAM NAME [COOKIE] [ENCODING]>
<@CGIPATH>
<@CHAR CODE [ENCODING]>
<@CIPHER ACTION TYPE STR [KEY] [ENCODING]>
<@COL [NUM] [FORMAT] [ENCODING]>
<@COLS></@COLS>
<@COLUMN NAME [FORMAT] [ENCODING]>
<@COMMENT></@COMMENT>
<@CONTINUE>
<@CURCOL>
<@CURRENTACTION [ENCODING]>
<@CURRENTDATE [FORMAT] [ENCODING]>
<@CURRENTTIME [FORMAT] [ENCODING]>
<@CURRENTTIMESTAMP [FORMAT] [ENCODING]>
<@CURROW>
<@DATEDIFF DATE1 DATE2 [FORMAT]>
<@DATETOSECS DATE [FORMAT]>
<@DAYS DATE DAYS [FORMAT] [ENCODING]>
<@DBMS [ENCODING]>
<@DEBUG></@DEBUG>
<@DELROWS ARRAY [POSITION] [NUM] [SCOPE]>
<@DISTINCT ARRAY [COLS] [SCOPE]>
<@DOCS FILE [ENCODING]>
<@DQ>
<@DSDATE DATE [INFORMAT] [ENCODING]>
<@DSTIME TIME [INFORMAT] [ENCODING]>
<@DSTIMESTAMP TS [INFORMAT] [ENCODING]>
<@DSNUM NUM [ENCODING]>

<@DSTYPE [ENCODING]>
<@ERROR PART [ENCODING]>
<@ERRORS></@ERRORS>
<@EXCLUDE></@EXCLUDE>
<@EXIT>
<@FILTER ARRAY EXPR [SCOPE]>
<@FOR [START] [STOP] [STEP] [PUSH]></@FOR>
<@FORMAT STR [FORMAT] [INFORMAT] [ENCODING]>
<@IF EXPR [TRUE] [FALSE]>
<@IF EXPR>
[<@ELSEIF EXPR>]
[<@ELSIFEMPTY VALUE>]
[<@ELSEIFEQUAL VALUE1 VALUE2>]
[<@ELSE>]
</@IF>
<@IFEMPTY VALUE>
[<@ELSE>]
</@IF>
<@IFEQUAL VALUE1 VALUE2>
[<@ELSE>]
</@IF>
<@INCLUDE FILE>
<@INTERSECT ARRAY1 ARRAY2 [COLS] [SCOPE1] [SCOPE2]>
<@ISDATE VALUE>
<@ISTIME VALUE>
<@ISTIMESTAMP VALUE>
<@ISNUM VALUE>
<@KEEP STR CHARS [ENCODING]>
<@LEFT STR NUMCHARS [ENCODING]>
<@LENGTH STR>
<@LITERAL VALUE [ENCODING]>
<@LOCATE STR FINDSTR>
<@LOWER STR [ENCODING]>
<@LTRIM STR [ENCODING]>
<@MAXROWS>
<@NEXTVAL NAME [SCOPE] [STEP]>
<@NUMAFFECTED>
<@NUMCOLS [ARRAY]>
<@NUMROWS [ARRAY]>
<@OMIT STR CHARS [ENCODING]>
<@PLATFORM [ENCODING]>
<@PRODUCT [ENCODING]>
<@POSTARG NAME [TYPE] [FORMAT] [ENCODING]>
<@POSTARGNAMES>
<@PURGE [NAME] [SCOPE]>
<@PURGERESULTS>
<@RANDOM [HIGH] [LOW]>

<@REGEX EXPR STR TYPE>
<@REPLACE STR FINDSTR REPLACESTR [POSITION] [ENCODING]>
<@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] [FORMAT] [ENCODING]>
<@SEARCHARGNAMES>
<@SECSTODATE SECS [FORMAT] [ENCODING]>
<@SECSTOTIME SECS [FORMAT] [ENCODING]>
<@SECSTOTS SECS [FORMAT] [ENCODING]>
<@SERVERSTATUS [VALUE] [ENCODING]>
<@SETCOOKIES>
<@SORT ARRAY [COLS] [SCOPE]>
<@SQ>
<@SQL [ENCODING]>
<@STARTROW>
<@SUBSTRING STR START NUMCHARS [ENCODING]>
<@TIMER [NAME] [VALUE]>
<@TIMETOSECS TIME [FORMAT]>
<@TMPFILENAME [ENCODING]>
<@TOGMT TS [FORMAT] [ENCODING]>
<@TOKENIZE VALUE CHARS>
<@TOTALROWS>
<@TRANSPOSE ARRAY [SCOPE]>
<@TRIM STR [ENCODING]>
<@TSTOSECS TS [FORMAT]>
<@UNION ARRAY1 ARRAY2 [COLS] [SCOPE1] [SCOPE2]>
<@UPPER STR [ENCODING]>
<@URL LOCATION [BASE] [USERAGENT] [FROM] [ENCODING]>
<@URLENCODE STR>
<@USERREFERENCE>
<@USERREFERENCEARGUMENT>
<@USERREFERENCECOOKIE>
<@VAR NAME [SCOPE] [FORMAT] [TYPE] [APREFIX] [ASUFFIX]
[RPREFIX] [RSUFFIX] [CPREFIX] [CSUFFIX] [ENCODING]>
<@VARINFO NAME ATTRIBUTE [SCOPE]>
<@VARNAMES SCOPE>
<@VERSION [ENCODING]>
<@! COMMENT>

Meta Tags List by Function

URL, Form, and CGI Values

@ARG
@ARGNAMES
@CGI
@CGIPARAM
@CGIPATH
@POSTARG
@POSTARGNAMES
@SEARCHARG
@SEARCHARGNAMES
@URL
@URLENCODE

Conditionals

@ELSE
@ELSEIF
@ELSEIFEMPTY
@ELSEIFEQUAL
@IF
@IFEMPTY
@IFEQUAL
@ISNUM

Paths

@APPFILE
@APPFILEPATH
@CGI
@CGIPARAM
@CGIPATH

Variables

@ASSIGN
@LITERAL
@PURGE
@USERREFERENCE
@USERREFERENCEARGUMENT
@USERREFERENCECOOKIE
@VAR
@VARINFO
@VARNAMES

Tango Information

@PLATFORM
@PRODUCT
@VERSION

File Access

@APPFILE
@APPFILENAME
@APPFILEPATH
@INCLUDE
@TMPFILENAME

Action/Application File Information

@ACTIONRESULT
@APPFILENAME
@CURRENTACTION
@RESULTS
@SQL

String Operations

@ASCII
@CHAR
@CIPHER
@DQ
@KEEP
@LEFT
@LENGTH
@LOCATE
@LOWER
@LTRIM
@OMIT
@REGEX
@REPLACE
@RIGHT
@RTRIM
@SQ
@SUBSTRING
@TOKENIZE
@TRIM
@UPPER
@URLENCODE

Numeric Operations

@CALC
@DSNUM
@ISNUM
@NEXTVAL
@RANDOM

Database Output

@ABSROW
@COL
@COLS
@COLUMN
@CURCOL
@CURROW
@FORMAT
@MAXROWS
@NUMAFFECTED
@NUMROWS
@PURGERESULTS
@ROWS
@STARTROW
@TOTALROWS

Data Source Information

@DBMS
@DSNUM
@DSTYPE
@SQL

Date and Time

@CURRENTDATE
@CURRENTTIME
@CURRENTTIMESTAMP
@DATEDIFF
@DATETOSECS
@DAYS
@ISDATE
@ISTIME
@ISTIMESTAMP
@SECSTODATE
@SECSTOTIME
@SECSTOTS
@TIMER
@TIMETOSECS
@TOGMT
@TSTOSECS

Server

@SERVERSTATUS

Array Operations

@ADDROWS
@ARRAY
@ASSIGN
@DELROWS
@DISTINCT
@FILTER
@INTERSECT
@NUMCOLS
@REGEX
@ROWS
@SORT
@TOKENIZE
@TRANSPOSE
@UNION
@VAR
@VARINFO

Data Validation

@ISDATE
@ISNUM
@ISTIME
@ISTIMESTAMP

Formatting

@FORMAT

Script Execution

@SCRIPT

Loop Processing

@BREAK
@CONTINUE

HTML Processing

@COMMENT
@EXCLUDE
@EXIT
@!

<@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 75
<@MAXROWS> page 127
<@ROWS> </@ROWS> page 145
<@STARTROW> page 162
<@TOTALROWS> page 169

<@ACTIONRESULT>

Syntax

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

Description

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.

Note For FileMaker Pro data sources (Mac OS), Insert actions return a single item of data—the record ID of the newly created record—that may be accessed by <@ACTIONRESULT *InsertActionName* 1>.

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 66
<@COLUMN>	page 68
Encoding Attribute	page 10
<@FORMAT>	page 102
Format Attribute	page 13
<@PURGERESULTS>	page 138
<@RESULTS>	page 143

<@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.

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 local variable colors contains the following array:

orange
amber
burnt umber

and the local variable colors2 contains the following array:

yellow

<@ADDROWS ARRAY="colors" SCOPE="local" VALUE="@@local$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 82
<@UNION> page 174

<@APPFILE>

Syntax

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

Description

Returns the path to the current application file, including the file name. 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 one of the following:

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

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 34
<@CGI>	page 57
Encoding Attribute	page 10
<@URLENCODE>	page 180

<@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">
<@URLENCODE STR="<@APPFILE>">
```

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 32
<@APPFILEPATH> page 34
<@INCLUDE> page 110
<@URLENCODE> page 180

<@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.

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 32
<@CGI>	page 57
Encoding Attribute	page 10
<@INCLUDE>	page 110
<@URLENCODE>	page 180

<@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 Tango.

The optional FORMAT attribute determines how the value is formatted by Tango. This attribute 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">
```

This example returns an array containing all values for the “foo” argument.

See Also

Encoding Attribute	page 10
Format Attribute	page 13
<@POSTARG>	page 134
<@SEARCHARG>	page 151

<@ARGNAMES>

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.

Example

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

This would return something like:

```
Fred          POST
access        POST
username      SEARCH
```

See Also

<@POSTARGNAMES> page 135
<@SEARCHARGNAMES> page 152

<@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 197 and “rDelim” on page 214.

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.

Examples

Creating an array and assigning it to a variable:

```
<@ASSIGN NAME="array1" VALUE="<@ARRAY ROWS='5'
COLS='3'>">
```

Creating and initializing an array, assigning it to a variable, and printing it:

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

```
<@ASSIGNNAME="array2"VALUE="<@ARRAYROWS='5'COLS='3'
VALUE=@@initValue CDELIM=',' RDELIM=';'>">
```

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

See Also

<@ASSIGN>	page 41
<@VAR>	page 184

<@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 Tango Server is running.

The attribute may be a literal value or a meta tag that returns a string.

Examples

```
<@ASCII CHAR="T">
```

This example returns “84”.

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

This example returns the ASCII value of the first character of the CHAR form field entered by the user.

See Also

<@CHAR> page 62

<@ASSIGN>

Syntax

<@ASSIGN NAME=*name* VALUE=*value* [SCOPE=*myscope*]>

Description

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

For more details on variables, see Chapter 7, "Using Variables," in the *User's Guide*.

The value may be text or an array. 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.

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
- 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.

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 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 "*".

The SCOPE attribute specifies the name of the scope in which to assign the variable. If this attribute is omitted, the following steps

are taken to determine the scope in which the assignment takes place:

- Tango searches for the variable in local, user, domain, and system scope, in that order. As soon as a 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 user, but can be changed by setting the defaultScope configuration variable.

For more information, see “defaultScope” on page 206.

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 local, user, domain or system scope, depending on the first instance of foo2 that Tango 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="foo3" SCOPE="user" VALUE="<@ARRAY
ROWS=5 COLS=3>">
```

This example assigns an 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 an array variable called array2, initializes it, 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="local">
```

Assigns to the local 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 38
<@PURGE>	page 137
<@VAR>	page 184

<@BREAK>

Description

Terminates execution of an <@COLS>, <@ROWS>, or <@FOR> 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

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

```
<@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.
```

See Also

<@COLS> </@COLS> page 67
<@CONTINUE> page 70
<@EXIT> page 97
<@FOR> </@FOR> page 101
<@ROWS> </@ROWS> page 145

<@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 Tango 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
```

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

For more information, see "currencyChar" on page 199, "decimalChar" on page 204, "DBDecimalChar" on page 203, and "thousandsChar" on page 218.

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(01238 0)`	*ERROR: digit 8 is invalid*
`num(123)`	*ERROR: digits 2 and 3 are invalid*

Strings

Any Tango 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 55.

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 49.

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 52.

`<@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.

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*)
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*)

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*)

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*...)
mean	$\boldsymbol{A_{mean}=(A_1+A_2+...+A_n)\ /n}$. returns the mean of the elements	mean(*expr expr*...)
var	$\boldsymbol{A_{var}=((A_1-A_{mean})^2+((A_2-A_{mean})^2+...+((A_n-A_{mean})^2)/(n-1)}$ 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 38.

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 41.

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 51). 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 38
`<@ELSEIF>`	page 103
`<@ELSEIFEMPTY>`	page 103
`<@ELSEIFEQUAL>`	page 103
Encoding Attribute	page 10
`<@FORMAT>`	page 102
Format Attribute	page 13
`<@IF>, <@ELSE>`	page 103
`<@VAR>`	page 184

<@CGI>

Syntax

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

Description

Returns the full path and name of the Tango CGI. On Mac OS, the path arguments separator, "$", is appended to this value.

With server plug-in / extension versions of Tango, this meta tag returns nothing.

Use this meta tag when creating embedded links to other Tango application files. Doing so ensures that the links work regardless of the Web server setup, or on which platform you are running Tango 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 one of the following:

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

Examples

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

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

If running the Mac OS CGI in the Web server's root directory, the example returns `/Tango.acgi$/custlist.taf` after the meta tag is evaluated. On Windows NT, with the CGI in the `cgi-bin` directory, the example returns `/cgi-bin/t3cgi.exe/custlist.taf`. If you are running Tango 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 32
<@APPFILEPATH>	page 34
Encoding Attribute	page 10

<@CGIPARAM>

Syntax

`<@CGIPARAM NAME=`*name* `[COOKIE=`*cookie*`] [ENCODING=`*encoding*`]>`

Description

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

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

Attribute Name	Description
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_TYPE	The MIME type of the HTTP request contents.
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, <@CGIPARAM NAME="HTTP_COOKIE" COOKIE="SICode"> returns the value of the SICode cookie. (This attribute is retained for backwards compatibility with Tango 2.3. It is recommended that you use <@VAR> with SCOPE="COOKIE" to return the values of cookies in Tango 3.0. See "<@VAR>" on page 184.)
HTTP_SEARCH_ARGS	Text after question mark (?) in the URL.
METHOD	The HTTP request method used for the current request. If a normal URL call, or form submitted with the GET method, "GET"; if a form submitted with the POST method, "POST".
PATH_ARGS	Text after the Tango CGI name and before any search arguments in the URL. If using the plug-in, or suffix-mapping to call the CGI automatically, no value is returned by this attribute. You may also use the <@APPFILE> meta tag to get this value (it works regardless of calling method and whether you use the plug-in or CGI). For example, a "/" after the CGI name indicates the beginning of the path args.

Attribute Name	Description
POST_ARGS	The raw POST (form submission) argument contents, containing the names and values of all form fields.
REFERER	The URL of the 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.
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.
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, Windows, etc.) on which the Web browser is running, and the application's version. For example, Internet Explorer 3.0 for Macintosh returns Mozilla/2.0 (compatible; MSIE 3.0; Mac_PowerPC).

The parameters listed in the following table are available under the Mac OS only. Some may be available only with the StarNine WebSTAR server. Check your server documentation to see if it supports a particular CGI parameter.

Parameter	Description
ACTION	The name of the Web server action used to call Tango.
ACTION_PATH	The path from the Web server application to the Tango CGI used to call the current application file. If using the plug-in, returns "Tango".
FULL_REQUEST	The entire contents of the current HTTP request, including the URL and any POST arguments.
PASSWORD	The password, obtained with HTTP authentication, of the user who requested the URL. This parameter is available only if the URL used to call the current application file is protected by a Web server realm.

Parameter	Description
SCRIPT_NAME	If using the plug-in, or if the Tango CGI is called with a Web server action, returns the application file path and name; otherwise, returns the path and name of the Tango CGI.

Example

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

See Also

Encoding Attribute page 10

<@CGIPATH>

Description

Mac OS Only

Returns the path to the directory containing the Tango CGI. The final slash is not included in the value. For example, if the Tango Server CGI is in a directory called Tango at the Web server root level, then `<@CGIPATH>` evaluates to `/Tango`.

If you have the Tango CGI at the root directory of the Web server, or are running the plug-in version of Tango Server, this meta tag returns an empty string.

This meta tag is supported mainly for backward compatibility with existing applications. For greater compatibility with the plug-in and other platform versions of Tango, Pervasive strongly recommends that the CGI, if being used, be located in the Web server's document root directory.

Note This meta tag returns a value only when using Tango CGI on the Mac OS. On other platforms, and for Tango Server plug-in on Mac OS, this meta tag evaluates to an empty string.

Example

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

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

See Also

`<@APPFILE>`	page 32
`<@APPFILEPATH>`	page 34
`<@CGI>`	page 57

<@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 Tango Server is running.

The number attribute may be a literal value or a meta tag that returns a number.

Examples

```
<@CHAR CODE=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.

```
<@OMIT STR=<@POSTARG NAME=comments> CHARS="<@CHAR
CODE=10><@CHAR CODE=9>">
```

This example returns the comments form field value stripped of any linefeed and tab characters.

See Also

<@ASCII> page 40
<@CRLF> page 71
<@DQ>, <@SQ> page 88
Encoding Attribute page 10

<@CIPHER>

Syntax

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

Description

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

<@CIPHER> provides the Tango 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 Tango 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 may be required or prohibited depending on the TYPE of cipher requested. Keys may be 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
```

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 table lists types of ciphers, their actions, their key restrictions, and a short description of each cipher.

Type	Action	Key Restrictions	Short Description
BitRoll	encrypt/decrypt	prohibited	swaps position of first 3 and last 5 bits in a byte
Caesar	encrypt/decrypt	optional, integer (positive and negative) values only, use “3” as default	rotate chars by value positions mod 26
OneTimePad	encrypt/decrypt	required, all alphabetic (no spaces or punctuation)	rotate characters by x positions, x being successive case-insensitive characters of key, a=1, b=2, ...
Rot13	encrypt/decrypt	prohibited	rotate characters by 13 positions
	hash	ignored	MD5* one way hash. Produces a 32 character string.

* ©RSA Data Security Inc. MD5 Message-Digest Algorithm.

The `ACTION` has two directions, forward and reverse. This means that you can take a string and encrypt, encipher or hash it in the forward direction, and, for the reverse direction, you can decrypt 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 two ciphering operations are supported:

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

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.

See Also

Encoding Attribute page 10

<@COL>

Syntax

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

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 valid only in a <@ROWS> block. This meta tag can be used with no attributes inside a <@COLS> block. In this case, it returns the value of the current column.

Note Insert actions using FileMaker Pro data sources (Mac OS) allow the use of <@COL 1> in the Results HTML. The meta tag evaluates to the record ID of the inserted record in this case.

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 67
<@COLUMN>	page 68
Encoding Attribute	page 10
<@FORMAT>	page 102
Format Attribute	page 13
<@ROWS> </@ROWS>	page 145

<@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

<@CURCOL> page 72
<@NUMCOLS> page 130

<@COLUMN>

Syntax

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

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.

Note For FileMaker Pro data sources (Mac OS), the name may be in field or layout.field format.

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 66
<@COLS> </@COLS> page 67
<@CURCOL> page 72
Encoding Attribute page 10
Format Attribute page 13

<@COMMENT> </@COMMENT>

Syntax

<@COMMENT>*comment*</@COMMENT>

Description

This tag pair gives you the ability to comment on Tango application files.

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.

<@CONTINUE>

Description

Terminates execution of the current iteration of an <@COLS>, <@ROWS>, or <@FOR> block. Execution of the loop continues from the beginning of the block. Outside of an <@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

The following 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).

```
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.
```

See Also

<@BREAK> page 44
<@COLS> </@COLS> page 67
<@EXIT> page 97
<@FOR> </@FOR> page 101
<@ROWS> </@ROWS> page 145

<@CRLF>

Description

Evaluates to a carriage return / linefeed combination.

This tag is used in the HTTP header specified by the `headerFile` configuration variable to generate the line terminators required for the HTTP header. (This tag is intended as a general-purpose header line terminator and may evaluate to a different character sequence to accommodate certain Web servers.)

You should use the `<@CRLF>` tag when editing the contents of the `header.htx` file.

See the chapter "Using Tango Server" in the *User's Guide* for more information on the Tango Server configuration file and how to change system configuration variables.

See Also

`headerFile` page 207
`<@LITERAL>` page 123

<@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

<@COLS> </@COLS> page 67
<@NUMCOLS> page 130

<@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 10

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

Syntax

<@CURRENTDATE [FORMAT=*format*] [ENCODING=*encoding*]>

<@CURRENTTIME [FORMAT=*format*] [ENCODING=*encoding*]>

<@CURRENTTIMESTAMP [FORMAT=*format*] [ENCODING=*encoding*]>

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 202.

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 Tango 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 201
Encoding Attribute	page 10
<@FORMAT>	page 102
Format Attribute	page 13
timeFormat	page 201
timestampFormat	page 201

<@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 28
<@NUMROWS>	page 131
<@ROWS> </@ROWS>	page 145

<@DATEDIFF>

Syntax

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

Description

Returns the number of days between the two dates specified.

For more information, see "<@ISDATE>, <@ISTIME>, <@ISTIMESTAMP>" on page 114.

<@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.

Example

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

This tag returns "7", the number of days between the two dates.

See Also

<@DAYS>	page 79
<@FORMAT>	page 102
Format Attribute	page 13
<@ISDATE>	page 114
<@ISTIME>	page 114
<@ISTIMESTAMP>	page 114

<@DATETOSECS>, <@SECSTODATE>

Syntax

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

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

Description

<@DATEOSECS> 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). For dates before this, the return values are negative.

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

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

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!".

Examples

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

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

```
<@DATETOSECS DATE=1969-12-31>
```

This tag returns "-86400", the number of seconds since January 1, 1970.

```
<@SECSTODATE SECS=-86400>
```

This tag returns "1969-12-31", the date derived from the number of seconds. The example assumes a dateFormat of "%Y-%m-%d".

See Also

`dateFormat`	page 201
Encoding Attribute	page 10
`<@FORMAT>`	page 102
Format Attribute	page 13
`<@ISDATE>`	page 114
`<@ISTIME>`	page 114
`<@ISTIMESTAMP>`	page 114
`<@SECSTOTIME>`	page 165
`<@SECSTOTS>`	page 172
`timeFormat`	page 201
`timestampFormat`	page 201
`<@TIMETOSECS>`	page 165
`<@TSTOSECS>`	page 172

<@DAYS>

Syntax

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

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!".

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 201
<@DATEDIFF>	page 76
Encoding Attribute	page 10
<@FORMAT>	page 102
Format Attribute	page 13

<@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.

Tango Enterprise only: 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 92
Encoding Attribute page 10

<@DEBUG> </@DEBUG>

Syntax

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

Description

These paired tags provide the Tango user more power to debug application files. If debugging is on, Tango processes the text inside the <@DEBUG></@DEBUG> pair; otherwise, these tags and the content inside are stripped out of the application file before being sent to the server.

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.

<@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 local variable colors contains the following array:

orange
amber
red
burnt umber

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

The local 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 30

<@DISTINCT>

Syntax

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

Description

Returns an array containing the distinct, or unique, rows in the input array.

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.

Meta tags are permitted in any of the attributes.

Examples

If the local 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="local">` returns:

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

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

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

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

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

See Also

`<@FILTER>` page 98
`<@INTERSECT>` page 111
`<@SORT>` page 158
`<@UNION>` page 174

<@DOCS>

Syntax

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

Description

Displays the content of an application 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 "Tango Server Configuration File" in the chapter "Using Tango Server" in the *User's Guide*.

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.

For more information, see "Saving an Application File as Run-Only" in Chapter 3 of the *User's Guide*.

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 10

<@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), Tango 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).

These meta tags are valid only in actions associated with a data source.

Note These meta tags are not applicable to FileMaker Pro data sources (Mac OS) as the date and time string formats required for FileMaker Pro are determined by layout and system settings that may be unavailable to Tango.

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 201
Encoding Attribute	page 10
Format Attribute	page 13
`timeFormat`	page 201
`timestampFormat`	page 201

<@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), Tango 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.

This meta tag is not applicable to FileMaker Pro data sources (Mac OS) as the number formats required for FileMaker Pro are determined by layout and system settings that may be unavailable to Tango.

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 199
`DBDecimalChar`	page 203
`decimalChar`	page 204
`<@DSDATE>`	page 89
`<@DSTIME>`	page 89
`<@DSTIMESTAMP>`	page 89
Encoding Attribute	page 10
`thousandsChar`	page 218

<@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
DAM	Mac OS	Data Access Manager
FileMaker	Mac OS	FileMaker Pro
ODBC	All	ODBC
Oracle	All (Tango 2.1 or greater)	Native Oracle

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 80
Encoding Attribute page 10

<@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 205.

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.

Tango 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" (Tango 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.
POSITION	The name of the action that generated the error, if applicable.
NUMBER1	The main error number.
NUMBER2	The secondary error number.
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 205
Encoding Attribute page 10
`<@ERRORS> </@ERRORS>` page 95

<@ERRORS> </@ERRORS>

Description

If more than one error occurs during application file execution, Tango 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 205
<@ERROR> page 93

<@EXCLUDE> </@EXCLUDE>

Syntax

```
<@EXCLUDE>text</@EXCLUDE>
```

Description

Processes *string* for meta tags, without adding the results of that processing to the Results HTML.

Like the <@COMMENT></@COMMENT> tag, any text inside the start and end tags is stripped out and does not appears in the HTML sent on to the Web server. Unlike that tag pair, any meta tags encountered are executed as part of the application file, not ignored as they are within a comment.

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 HTML contained inside it are removed before the HTML is returned, and <@ASSIGN> is executed as part of the application file.

See Also

<@COMMENT> </@COMMENT> page 69

<@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

The following 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".

```
[...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>
```

See Also

<@BREAK> page 44
<@CONTINUE> page 70

<@FILTER>

Syntax

<@FILTER ARRAY=*arrayVarName* EXPR=*filterExpr* [SCOPE=*scope*]>

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.

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 local 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="local" EXPR="#3=ACTIVE">` returns:

3243	Acme Insurance	ACTIVE
1234	Cinetopia	ACTIVE

3243	Acme Insurance	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

<@DISTINCT>	page 84
<@INTERSECT>	page 111
<@SORT>	page 158
<@UNION>	page 174

<@FOR> </@FOR>

Syntax

```
<@FOR [START=start] [STOP=stop] [STEP=step]
[PUSH=push]>
</@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 allows the sending of data to the client after the specified number of iterations have taken place.

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

Example

```
<@FOR STOP="5">
This function does this <BR>
</@FOR>
```

This example outputs the following:

```
This function does this
This function does this
This function does this
This function does this
This function does this
```

<@FORMAT>

Syntax

<@FORMAT STR=*string* [FORMAT=*format*] [INFORMAT=*informat*] [ENCODING=*encoding*]>

Summary

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

To output the current date in ODBC / ISO style, purposely using a timestamp.

```
<@FORMAT STR="<@CURRENTTIMESTAMP>"
FORMAT="datetime:%Y-%m-%d" INFORMAT="datetime:<@VAR
NAME='timestampFormat'>">
```

To output a thousands-grouped integer value.

```
If a kilobyte is 1024 (2^10 bytes), then a megabyte
should be <@FORMAT STR=<@CALC EXPR="1024 * 1024">
FORMAT="num:comma-integer"> bytes.
```

See Also

Encoding Attribute page 10
Format Attribute page 13

<@IF>, <@ELSEIF>, <@ELSEIFEMPTY>, <@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]
[<@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 45.

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.

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.

For more information, see "<@IFEMPTY> <@ELSE> </@IF>" on page 107 and "<@IFEQUAL> <@ELSE> </@IF>" on page 108 for descriptions of how the <@ELSEIFEQUAL> and <@ELSEIFEMPTY> conditions are evaluated.

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. There is no limit to the nested if levels on UNIX or Windows platforms; however, on Macintosh, the nested if limit is 12 levels.

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 45
<@IFEMPTY>, <@ELSE> page 107
<@IFEQUAL>, <@ELSE> page 108

<@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.

Macintosh only: These tags may be nested up to 12 levels; beyond this limit, an error is returned.

Example

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

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

See Also

<@ELSEIF>	page 103
<@ELSEIFEMPTY>	page 103
<@ELSEIFEQUAL>	page 103
<@IF>,<@ELSE>	page 103
<@IFEQUAL>,<@ELSE>	page 108

<@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, Tango attempts to convert both values to numbers and perform a numeric comparison. If one or both values cannot be converted to numbers, Tango performs a string comparison.

The *trueSubstitutionText* and *falseSubstitutionText* may include other <@IF>, <@IFEMPTY>, and <@IFEQUAL>.

Macintosh only: These tags may be nested up to 12 levels; beyond this limit, an error is returned.

Examples

```
<@IFEQUAL VALUE1="<@CGIPARAM 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

`<@ELSEIF>`	page 103
`<@ELSEIFEMPTY>`	page 103
`<@ELSEIFEQUAL>`	page 103
`<@IF>, <@ELSE>`	page 103
`<@IFEMPTY>, <@ELSE>`	page 107

<@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 Tango 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 Tango 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]>
```

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.

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	*

* Tango 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>
```

For more information, see “Array evaluation” on page 47.

This is because an array value specified as an expression (in `<@CALC>` or `<@IF>`) returns the number of rows in that array.

See Also

`<@DISTINCT>`	page 84
`<@FILTER>`	page 98
`<@SORT>`	page 158
`<@UNION>`	page 174

<@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 Tango 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 Tango 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.

<@ISNUM>

Syntax

<@ISNUM VALUE=*number*>

Summary

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 199
decimalChar	page 204
<@ISDATE>	page 114
<@ISTIME>	page 114
<@ISTIMESTAMP>	page 114
thousandsChar	page 218

<@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 values.

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 10
<@OMIT> page 132

<@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 10
<@REPLACE>	page 142
<@RIGHT>	page 144
<@SUBSTRING>	page 163

<@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.

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 Tango 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 206
Encoding Attribute	page 10
userKey, altuserKey	page 221

<@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="Tango Enterprise" FINDSTR="tango">
```

This example evaluates to “0”, because an exact match of “tango”, including case, is not found in the source string.

```
<@LOCATE STR="<@LOWER STR='Tango Enterprise'>"
FINDSTR="tango">
```

This example evaluates to “1.”

<@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 10
<@UPPER> page 177

<@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 10
<@KEEP> page 120
<@OMIT> page 132
<@RTRIM> page 147
stripCHARs page 218
<@TRIM> page 171

<@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 131
<@STARTROW> page 162
<@TOTALROWS> page 169

<@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 scalar, or if the step is non-integer, <@NEXTVAL> evaluates to nothing, and an error is logged if LogLevel is greater than 0.

Scalars 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 209
<@VAR> page 184

<@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 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".

Notes

- 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 67
<@CURCOL> page 72
<@NUMROWS> page 131

<@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><@COLUMNNAME="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

<@MAXROWS> page 127
<@NUMCOLS> page 130
<@STARTROW> page 162
<@TOTALROWS> page 169

<@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”.

See Also

Encoding Attribute page 10
<@KEEP> page 120
<@LTRIM> page 126
<@RTRIM> page 147
<@TRIM> page 171

<@PLATFORM>

Syntax

`<@PLATFORM [ENCODING=`*encoding*`]>`

Description

Returns the name of the operating system on which Tango Server is currently running. You may want to use this tag in Branch actions to branch to different External actions based on the current Tango Server platform.

Example

For example, `<@PLATFORM>` may evaluate to one of the following:

- SunOS/5.5; sun4m
- Windows NT/4.0; Intel
- Mac OS/ 8.0.1; PPC
- IRIX16.2; IP22

See Also

Encoding Attribute page 10
`<@VERSION>` page 191

<@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 Tango.

The optional FORMAT and ENCODING attributes determine how the value is formatted by Tango. 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 35
Encoding Attribute	page 10
Format Attribute	page 13
<@POSTARGNAMES>	page 135
<@SEARCHARG>	page 151
<@SEARCHARGNAMES>	page 152

<@POSTARGNAMES>

Description

Returns an array containing the names of all post arguments.

Post arguments are passed to Tango 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.

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

<@ARG> page 35
<@ARGNAMES> page 37
<@SEARCHARGNAMES> page 152

<@PRODUCT>

Syntax

<@PRODUCT ENCODING=*encoding*>

Description

Returns the name of the server's product type.

Example

<@PRODUCT> on a licensed copy of Tango Enterprise returns one of the following:

```
Tango Enterprise Application Server
Tango Enterprise Developer Studio
```

See Also

<@PLATFORM> page 133
<@VERSION> page 191

<@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 Purging variables in the *cookie* scope does not cause the *Web browser* to forget a cookie. If you want to make a Web browser forget a cookie, you must set the expiry time to immediate, for example, “in -1 days” in the Properties dialog box for a cookie variable when assigning values to variables with an Assign action.

Examples

The following examples demonstrate how to remove variable types from various scopes:

```
<@PURGE NAME="foo" SCOPE="user">
<@PURGE NAME="foo" SCOPE="domain">
<@PURGE NAME="foo" SCOPE="local">
<@PURGE NAME="foo" SCOPE="cookie">
```

The following examples demonstrate how to remove all variables from a given scope:

```
<@PURGE SCOPE="user">
<@PURGE SCOPE="domain">
<@PURGE SCOPE="local">
```

See Also

<@ASSIGN> page 41
<@VAR> page 184

<@PURGERESULTS>

Description

Empties the currently accumulated Results HTML.

Note <@PURGERESULTS> can only clear results that have been accumulated in previous actions. It does not clear the accumulated results of the current action.

See Also

<@ACTIONRESULT> page 29
<@RESULTS> page 143

<@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 45

<@REGEX>

Syntax

<@REGEX EXPR=*expression* STR=*text* TYPE=*type*>

Description

Provides an interface to POSIX regular expression matching routines from inside Tango. 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

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“.

<@REPLACE>

Syntax

<@REPLACE STR=*string* FINDSTR=*findString* REPLACESTR=*replaceString* [POSITION=*position*] [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.

<@REPLACE> is case insensitive.

Examples

```
<@REPLACE STR="alpha" FINDSTR="a" REPLACESTR="u"
POSITION="2">
```

This example returns “alphu”, replacing the second occurrence of “a”.

```
<@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”.

See Also

Encoding Attribute	page 10
<@LEFT>	page 121
<@LOCATE>	page 124
<@REGEX>	page 140
<@REPLACE>	page 142
<@RIGHT>	page 144
<@SUBSTRING>	page 163

<@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" ENCODING="NONE">. If the variable is empty, you would branch to the normal processing to query the database.

See Also

<@ACTIONRESULT> page 29
Encoding Attribute page 10
<@PURGERESULTS> page 138

<@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 10
<@LEFT>	page 121
<@LOCATE>	page 124
<@REGEX>	page 140
<@REPLACE>	page 142
<@SUBSTRING>	page 163

<@ROWS> </@ROWS>

Syntax

```
<@ROWS [ARRAY=array] [SCOPE=scope] [PUSH=push]
[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 local 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 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 Tango. Sometimes, even though Tango 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 66
<@COLUMN>	page 68
<@MAXROWS>	page 127

<@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.

Examples

```
<@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 10
<@KEEP> page 120
<@LTRIM> page 126
<@OMIT> page 132
stripCHARs page 218
<@TRIM> page 171

<@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=scopeSpec]> your script here</@SCRIPT>

This is the long 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 Tango tags; those tags are substituted *prior* to script execution. In order for the script to be able to interact with the Tango environment, there are predefined object / methods that can be called from the script. They are explained on the following page.

The optional SCOPE attribute defines the lifetime of the objects and functions declared in the script, and is similar to the scope of variables. Only two scopes are supported: LOCAL and IMMED. The default scope is LOCAL, so anything defined in one script can be referenced in another script in the same file execution. The second scope, IMMED, 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.

Nesting of <@SCRIPT> blocks is not supported.

Usage Two: <@SCRIPT EXPR="your script here" [SCOPE=SCOPESPEC]>

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 Tango 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 Tango Server to allow scripts to interact with Tango in a controlled and meaningful way:

- **server**: object representing Tango Server.
- **getVariable(name)**: gets Tango 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 Tango variable, using default scoping rules, returns nothing.
- **setVariable(name, value, scope)**: as in the previous paragraph, but with defined scope.

Note Tango variables are accessed by value, not by reference. You must therefore use *setVariable* to update Tango with any changes you make to variable values. Also, because they are passed by value, getting large Tango arrays can consume a lot of memory because the entire array is duplicated inside of JavaScript.

Because Tango supports only two-dimensional arrays, it is an error to try to put a JavaScript array of more than two dimensions into a Tango variable.

For more information on the JavaScript capabilities of Tango, see the online help for JavaScript that is distributed with Tango (in the `Help` directory under the Tango 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 Tango 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 41
`<@VAR>`	page 184

<@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 Tango.

The optional FORMAT and ENCODING attributes determine how the value is formatted by Tango. 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 35
Encoding Attribute page 10
Format Attribute page 13
<@POSTARG> page 134

<@SEARCHARGNAMES>

Description

Returns an array containing the names of all search arguments.

Search arguments are passed to Tango 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.

Example

The following returns all search argument names using the default array formatting:

```
<@ASSIGN NAME="mysearchargs"
VALUE="<@SEARCHARGNAMES>">

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

See Also

<@ARG> page 35
<@ARGNAMES> page 37
<@POSTARGNAMES> page 135

<@SECSTODATE>, <@SECSTOTIME>, <@SECSTOTS>

See the following meta tags:

<@DATETOSECS>, <@SECSTODATE>page 77
<@TIMETOSECS>, <@SECSTOTIME>page 165
<@TSTOSECS>, <@SECSTOTS> page 172

<@SERVERSTATUS>

Syntax

<@SERVERSTATUS [VALUE=*value*] [ENCODING=*encoding*]>

Description

Returns status information on Tango 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.

Category	Description
Version	version of status information (not the server)
ProcessID	system process ID number
UpTime (min)	server running time (in minutes)
ActiveQryThr	number of threads marked in use
AvgQryProcTime	average time to process a request, in 1/60 second ticks
LstQryProcTime	time to process last request, in 1/60 second ticks
DataSrcCount	number of data source connections allocated
NumQryServed	number of requests served since server inception
MinQryProcTime	minimum request processing time so far, in 1/60 second ticks
MaxQryProcTime	maximum request processing time so far, in 1/60 second ticks
AvgQryReadTime	average time to read a prepare a request for processing, in 1/60 second ticks
AvgQryWriteTime	average time to return results to the user after processing, in 1/60 second ticks
NumAFRead	number of Tango application files read from disk, cache, or network
AvgAFReadTime	average amount of time taken to read an application file from disk, cache, or network, in 1/60 second ticks
AvgAFSize	average size of application file read from disk, cache, or network
QryThrCount	number of processing threads allocated

Category	Description
ActiveDataSrc	number of data sources marked in use
NumQrykilled	number of requests killed (timed out) since server inception
TotlTripRdBytes	total number of bytes read via network
TotlTripRdFiles	total number of files read via network
TotlTripRdTime	total amount of time taken to read all files read via network, in 1/60 second ticks
NumCachedDocs	number of application files currently in application file cache
NumCachedIncl	number of include files currently in include file cache
ProcessSize	current size of server process (bytes)
HeapSize	current amount of working memory consumed (bytes)
NumTripBadConn	number of network CGI-server connections failed
NumUsersShared	number of user references in the shared variable store
NumVarsShared	number of variables in the shared variable store
NumUsersLocal	number of user references in the local variable store
NumVarsLocal	number of variables in the local variable store

Notes

- `NumAFRead`, `AvgAFReadTime`, and `AvgAFSize` include cache reads, and therefore 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 UNIX 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.
- Because of the way that Tango Server works under Macintosh, the following status values return zero under Mac OS: `TotlTripRdBytes`, `TotlTripRdFiles`, `TotlTripRdTime`, `NumTripBadConn`, and `ProcessID`.

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 10

<@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 Tango. If you do not, the cookie scope does not work properly.

See Also

`headerFile` page 207

<@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 local variable test contains the following array:

4	example
2	is
7	sorting
3	an
5	of
1	here
6	array

`<@SORT ARRAY="test" SCOPE="local" COLS="1 NUM">` rewrites the local `test` variable as the following array:

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 84
`<@FILTER>`	page 98
`<@INTERSECT>`	page 111
`<@UNION>`	page 174

<@SQ>

See the following meta tag:

`<@DQ>, <@SQ>` page 88

<@SQL>

Syntax

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

Description

Returns last action-generated SQL.

This meta tag accesses your last action-generated SQL.

This meta tag is not valid for use with FileMaker Pro data sources.

See Also

Encoding Attribute page 10

<@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 28
<@CURROW>	page 75
<@MAXROWS>	page 127
<@NUMROWS>	page 131
<@ROWS> </@ROWS>	page 145
<@TOTALROWS>	page 169

<@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 ” 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 10
<@LEFT>	page 121
<@LOCATE>	page 124
<@REPLACE>	page 142
<@RIGHT>	page 144

<@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.

Values returned by <@TIMER> are accurate to 1/60 of a second.

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>" on page 114.

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 10
<@FORMAT>	page 102
Format Attribute	page 13
<@ISDATE>	page 114
<@ISTIME>	page 114
<@ISTIMESTAMP>	page 114
timeFormat	page 219

<@TMPFILENAME>

Syntax

`<@TMPFILENAME [ENCODING=`*encoding*`]>`

Description

Generates a unique temporary file name on the file system that Tango 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 Tango Server.

See Also

Encoding Attribute page 10

<@TOGMT>

Syntax

<@TOGMT TS=*timestamp* [FORMAT=*format*] [ENCODING=*encoding*]>

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 Toronto, Ontario, the regular difference is 5 hours, but the summertime difference is 4 hours. Tango accounts for daylight savings time.

Example

```
<@TOGMT TS="<@CURRENTTIMESTAMP>">
```

See Also

<@CURRENTDATE>	page 74
<@CURRENTTIME>	page 74
<@CURRENTTIMESTAMP>	page 74
Encoding Attribute	page 10
<@FORMAT>	page 102
Format Attribute	page 13

<@TOKENIZE>

Syntax

<@TOKENIZE VALUE=*text* CHARS=*delimiters*>

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 VALUE of the text and the delimiting CHARS, and returns its results as an array. The result is a one row array, with a column for each token. If the entire string consisted of only delimiters, a one by one empty array is returned.

Each character in CHARS is taken as a separate delimiter.

Example

Note There are extra spaces in the string specified in the VALUE attribute in the following example.

```
<@TOKENIZE VALUE="  The  quick brown fox jumped,
quite amazingly, over the lazy dog. " CHARS=" ,.">
```

The array returned would look like this:

The	quick	brown	fox	jumped	quite	amazingly	over	the	lazy	dog

See Also

<@LTRIM> page 126
<@RTRIM> page 147
<@SUBSTRING> page 163

<@TOTALROWS>

Description

Returns the total number of rows matching the criteria specified in the Search action the meta tag is used in. The actual number of rows returned by the Search action is determined by the **Maximum Matches** and **Start Row** settings.

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. If this option is not selected, or if the tag is used outside of a Search action, this tag returns "1".

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 28
<@CURROW>	page 75
<@MAXROWS>	page 127
<@NUMROWS>	page 131
<@ROWS> </@ROWS>	page 145
<@STARTROW>	page 162

<@TRANSPOSE>

Syntax

<@TRANSPOSE ARRAY=*arrayVarName* [SCOPE=*scope*]>

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.

Example

If the local 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="local">'>
```

@@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 41

<@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 10
<@KEEP>	page 120
<@LTRIM>	page 126
<@OMIT>	page 132
<@RTRIM>	page 147

<@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. For dates before this, the return values are negative.

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="1969-12-31 23:59:55">
```

This example returns "-5".

```
<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 77
Encoding Attribute	page 10
<@FORMAT>	page 102
Format Attribute	page 13
<@ISDATE>	page 114
<@ISTIME>	page 114
<@ISTIMESTAMP>	page 114
<@SECSTODATE>	page 77
<@SECSTOTIME>	page 165
timestampFormat	page 201
<@TIMETOSECS>	page 165

<@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

* Tango 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 30
`<@DISTINCT>` page 84
`<@FILTER>` page 98
`<@INTERSECT>` page 111
`<@SORT>` page 158

<@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 10
<@LOWER> page 125

<@URL>

Syntax

<@URL LOCATION=*location* [BASE=*base*] [USERAGENT=*user-agent*] [FROM=*from*] [ENCODING=*encoding*]>

Description

Retrieves the specified URL and returns its data, stripped of the HTTP header.

<@URL> supports only HTTP URLs. The HTTP-type URL must be of the following form:

```
http://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. Tango prepends the specified value of the BASE attribute to the relative path.

For more information, see “userAgent” on page 220.

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.

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. (Tango 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 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 Tango 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.

For more information, see the section "Timed URL Processing With Tango Server" in the chapter "Using Tango Sever" in the *User's Guide*.

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 Tango's timed URL processing functionality. If something goes wrong, the destination server administrator knows who to contact.

(Tango application files can get the FROM information from a request using <@CGIPARAM NAME="FROM_USER">.)

If you do not specify a value, the default value is given by the configuration variable, mailDefaultFrom.

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, you must use the ENCODING=NONE attribute.

Note <@URL> currently has some limitations. It does not support:

- the HTTPS protocol for secure access to Web sites
- access to user / password protected Web sites
- the sending of post arguments (form fields) with a request.

Examples

```
<@URL LOCATION="http://www.redhat.com/">
```

```
<@URL LOCATION="http://www.redhat.com/"
BASE="http://www.redhat.com/"
USERAGENT="Mozilla/4.04 [en] (WinNT; I)"
FROM="tango-admin@mycompany.com">
```

See Also

Encoding Attribute	page 10
mailDefaultFrom	page 210
userAgent	page 220

<@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 178

<@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.

For more information on user variable tracking, see Chapter 7, "Using Variables," in the *User's Guide*.

The user reference number can be used for reliable tracking of user variables.

See Also

`userKey, altuserKey`	page 221
`<@USERREFERENCEARGUMENT>`	page 182
`<@USERREFERENCECOOKIE>`	page 183

<@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.

For more information on user variable tracking, see Chapter 7, "Using Variables," in the *User's Guide*.

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 221
`<@USERREFERENCE>`	page 181
`<@USERREFERENCECOOKIE>`	page 183

<@USERREFERENCECOOKIE>

Description

Used in the default HTTP header of Tango when returning results. It permits intelligent setting of the user reference cookie, a value that can be used to track user variables.

For more information on user variable tracking, see Chapter 7, "Using Variables," in the *User's Guide*.

If no Tango user reference number was received, either via cookie or search argument, with the current HTTP request, <@USERREFERENCECOOKIE> returns the following:

```
Set-Cookie: Tango_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 Tango's default HTTP header for Mac OS versions:

```
HTTP/1.0 200 OK [CRLF]
Server: WebSTAR/1.0 ID/ACGI [CRLF]
MIME-Version: 1.0 [CRLF]
Content-type: text/html [CRLF]
<@USERREFERENCECOOKIE>[CRLF]
```

See Also

userKey, altuserKey	page 221
<@USERREFERENCE>	page 181
<@USERREFERENCEARGUMENT>	page 182

<@VAR>

Syntax

```
<@VAR NAME=name [SCOPE=scope] [FORMAT=format]
[APREFIX=aprefix] [ASUFFIX=asuffix] [RPREFIX=rprefix]
[RSUFFIX=rsuffix] [CPREFIX=cprefix] [CSUFFIX=csuffix]
[TYPE=text] [ENCODING=encoding]>
```

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.

Scalars

When retrieving the contents of a scalar (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).

The format outputted is specified by the following attributes:

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

For more information, see “aPrefix” on page 196, “aSuffix” on page 196, “cPrefix” on page 198, “cSuffix” on page 199, “rPrefix” on page 215 and “rSuffix” on page 215.

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 or ordered lists. The default values of these attributes are set by configuration variables with the same name.

Scoping Rules

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 193.

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

For more information, see “domainScopeKey” on page 206.

- **Domain Scope** contains variables that a class of users can share. These variables can be accessed from any application file run by any user within the class. This scope is defined by setting the system configuration variable `domainScopeKey` appropriately; that is, setting it to a value that can differentiate such users. To use this scope, specify `SCOPE=`*domain*.

For more information, see “userKey, altuserKey” on page 221.

- **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*.
- **Local 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=`*local*, or `SCOPE=`*doc*.
- **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*.

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.

Behavior is undefined when both methods are used at once.

Scoping Precedence

When no scope is specified, Tango must find the variable by looking for the variable name within the various scopes. Tango has a set order in which it tries to find scopes. They are:

```
local—»user—»domain—»system
```

Note Variable scoping precedence for variables and configuration variables does not check cookie scope.

For more information, see “domainScopeKey” on page 206.

If `domainScopeKey` resolves to empty for this user, then domain is not checked.

Variable Shortcut Syntax

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
```

Configuration Variables

For a detailed list of configuration variables, see Chapter 2 of the *Meta Tags and Configuration Variables* manual.

Tango reserves special variables that contribute to the configuration of the server and also that provide default behaviors for users.

For more information, see “About Variables” in Chapter 7 of the *User’s Guide*. For more information on setting configuration variables using the `config.taf` file, see “Tango Server Configuration File” in the *User’s Guide*.

System variables that contribute to the basic configuration of the server only exist in the system scope. System variables that provide default behaviors for users 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 local variable:

```
<@VAR NAME="foo" SCOPE="local">
<@VAR NAME="local$foo">
@@local$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:

```
<@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>">
```

See Also

<@ARRAY>	page 38
<@ASSIGN>	page 41
Encoding Attribute	page 10
Format Attribute	page 13

<@VARINFO>

Syntax

<@VARINFO NAME=*variable* ATTRIBUTE=*attribute* [SCOPE=*scope*]>

Description

Returns information about variables and accepts three ATTRIBUTE values, TYPE, ROWS, and COLS:

- TYPE returns either text or array.
- 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="scalar" SCOPE="user" VALUE="abcdef">
<@ASSIGN NAME="array" SCOPE=user" VALUE="<@ARRAY
ROWS='5' COLS='3'>">
```

<@VARINFO> returns the following values:

```
<@VARINFO NAME="scalar" SCOPE="user" ATTRIBUTE="type">
        (returns "text")
<@VARINFO NAME="scalar" SCOPE="user" ATTRIBUTE="rows">
        (returns "0")
<@VARINFO NAME="scalar" 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 41
<@VAR> page 184
<@VARNAMES> page 190

<@VARNAMES>

Syntax

<@VARNAMES SCOPE=*scope*>

Description

Returns an array containing all variable names for a given scope. See <@VAR> for an explanation of the scoping rules. The result array has one column and *n* rows where *n* is the number of variables in the specified scope.

Example

The following returns all variable names for the current user scope using the default array formatting:

```
<@ASSIGN NAME="myvarnames" VALUE="<@VARNAMES
SCOPE='user'>">
<@VAR NAME="myvarnames">
```

See Also

<@ASSIGN> page 41
<@VAR> page 184

<@VERSION>

Syntax `<@VERSION [ENCODING=`*encoding*`]>`

Description Returns the version number of Tango Server.

Example `<@VERSION>` returns the version number of Tango Server, for example, “3.0.012”.

See Also

Encoding Attribute page 10
`<@PLATFORM>` page 133

<@!>

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” on page 8; 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 69
<@EXCLUDE> </@EXCLUDE> page 96

CHAPTER 2

Configuration Variables

2

Setting Tango Options With Configuration Variables

Configuration variables set options controlling the operation of Tango 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. See "Working With Variables" in Chapter 7 of the *User's Guide* for details of variable assignment.

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. See the chapter "Using Tango Server" in the *User's Guide*, and the *Getting Started Guide* for details.

Tango Server switches (for example, `fileReadSwitch`, `javaSwitch`) are special configuration variables that enable or disable certain Tango features. Documentation on Tango Server switches is provided in the *User's Guide*: see "Feature Switches" in the chapter "Using Tango Server" for details on these configuration variables.

Configuration variables are saved in the configuration file (`t3server.ini` on Windows and UNIX and `Tango 3 Server Preferences` under Mac OS). See the chapter "Using Tango Server" in the *User's Guide* for more details about this file.

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”).

Note Configuration variables are never valid in cookie scope.

For more information, see “Understanding Scope” in Chapter 7 of the *User’s Guide*.

For those variables that belong to all or a variety of scopes, adding scope specifications has the following effects:

- `scope=LOCAL` 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=DOMAIN` sets the configuration variable value to be used in the current domain.

For more information, see the chapter “Using Tango Server” in the *User’s Guide*.

- `scope=SYSTEM` sets the configuration variable value to be used in Tango 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 Tango configuration variables.

- **Mac OS**
  StartupDisk:System Folder:Preferences:Tango3
- **Windows NT**
  C:\WinNT\Tango3\
- **Windows 95**
  C:\Windows\Tango3\

Note Under Windows, the drive letter may be different than "C", depending on where Windows is installed.

- **UNIX**
  /var/opt/EDI/

This affects the following configuration variables:

defaultErrorFile	page 205
headerFile	page 207
logDir	page 209
pidFile	page 213
timeoutHTML	page 220
varCachePath	page 223

altUserKey

See “userKey, altuserKey” on page 221.

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).

For more information, see “Arrays” in Chapter 7 of the *User’s Guide*.

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 196
cPrefix	page 198
cSuffix	page 199
rPrefix	page 215
rSuffix	page 215

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).

For more information, see “Arrays” in Chapter 7 of the *User’s Guide*.

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 196
cPrefix	page 198
cSuffix	page 199
rPrefix	page 215
rSuffix	page 215

cache

System scope only

This configuration variable turns the Tango Server cache on or off. Possible values are `true` and `false`.

The default value of this variable is `true`.

See Also

`cacheSize` page 197

cacheSize

System scope only

The value of this configuration variable specifies, in bytes, how much of Tango memory is used for caching application files and files referenced with `<@INCLUDE>`.

Tango 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 Tango 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 Tango.

The default value of `cacheSize` is `2000000`.

See Also

`cache` page 197

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 “`,`”.

For more information, see “Arrays” in Chapter 7 of the *User’s Guide*.

See Also

`rDelim` page 214

configPasswd

User 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, Tango checks to see if there is a user variable called `configPasswd` that matches the corresponding system variable. If there is, Tango lets you change configuration variables. If not, Tango returns an error message.

For more information, see the chapter "Using Tango Server" in the *User's Guide*.

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 `config.taf` application file, you are prompted for this password.

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).

For more information, see "Arrays" in Chapter 7 of the *User's Guide*.

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 196
`aSuffix`	page 196
`cSuffix`	page 199
`rPrefix`	page 215
`rSuffix`	page 215

crontabFile

System scope only

This configuration variable points to the `crontab` file used to set up timed URL processing with Tango.

The default value of this configuration variable is empty.

For more information, see "Timed URL Processing With Tango Server" in the chapter "Using Tango Server" in the *User's Guide*.

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.

For more information, see "Arrays" in Chapter 7 of the *User's Guide*.

See Also

`aPrefix`	page 196
`aSuffix`	page 196
`cPrefix`	page 198
`rPrefix`	page 215
`rSuffix`	page 215

currencyChar

Valid in all scopes

(local scope invalid when staticNumericChars=true)

The value of this configuration variable tells Tango 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.

Tango 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 Tango Server is constructing SQL for Search, Insert, Update, and Delete actions. Tango 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 Mac OS, 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 217.

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 local scope have no effect.

See "Understanding Scope" in Chapter 7 of the *User's Guide*.

When staticNumericChars has the value FALSE, currencyChar works with scope in the standard way.

See Also

DBDecimalChar	page 203
decimalChar	page 204
<@DSDATE>	page 89
<@DSNUM>	page 91
<@DSTIME>	page 89
<@DSTIMESTAMP>	page 89
staticNumericChars	page 217
thousandsChar	page 218

dataSourceLife

System scope only

Tango Enterprise only.

The value of this configuration variable indicates how long the Tango 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

Tango Enterprise only: 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. Tango converts these values to the formats required by the database.

Note On Mac OS, 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 Tango.

FileMaker Pro data sources only: These configuration variables determine how the date, time, and timestamp values returned by the `<@CURRENTDATE>`, `<@CURRENTTIME>`, and `<@CURRENTTIMESTAMP>` meta tags are formatted. Date, time, and timestamp values specified in Update and Insert actions, and those in criteria values must match the format specified in the *Date & Time* control panel of the Macintosh running the FileMaker Pro application.

For more information, see “DATETIME” on page 16.

To control the format of dates and times returned by FileMaker Pro, use the `FORMAT` attribute with the `datetime:` class of formatting.

The following table shows valid formatting codes.

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 as first day of week) (00–53)
%w	weekday (0–6, Sunday is zero)
%W	week number of the year (Monday as 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”

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 Tango 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 Tango Server.

If you use ODBC data sources requiring different decimal characters, you may use the `DBDecimalChar` with `scope=LOCAL` to change this setting temporarily while accessing a specific data source.

The setting of `DBDecimalChar` is used when Tango 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`—Tango 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 199
decimalChar page 204
thousandsChar page 218
<@DSDATE> page 89
<@DSTIME> page 89
<@DSTIMESTAMP> page 89
<@DSNUM> page 91

debugMode

Valid in all scopes

For more information, see “Debugging Application Files” in Chapter 3 of the *User’s Guide*.

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).

decimalChar

Valid in all scopes

(local scope invalid when staticNumericChars=true)

The value of this configuration variable tells Tango 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.

Tango 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 Tango Server is constructing SQL for Search, Insert, Update, and Delete actions. If necessary, Tango 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.

Tango uses this setting to format any numeric values retrieved from a data source.

The default value of `decimalChar` is "." (a period).

On Mac OS, 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 217.

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 local scope have no effect.

See "Understanding Scope" in Chapter 7 of the *User's Guide*.

When `staticNumericChars` has the value `FALSE`, `decimalChar` works with scope in the standard way.

See Also

`currencyChar`	page 199
`DBDecimalChar`	page 203
`<@DSDATE>`	page 89
`<@DSNUM>`	page 91
`<@DSTIME>`	page 89
`<@DSTIMESTAMP>`	page 89
`staticNumericChars`	page 217
`thousandsChar`	page 218

defaultErrorFile

System scope only

This configuration variable specifies a path to a file on the Tango Server machine. Tango 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.

defaultScope

Local 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 user.

domainScopeKey

System scope only

This configuration variable sets the key for the domain scope; that is, what value Tango uses in order to determine which domain a request originated from and the value it uses as a key to find domain variables internally.

The default value is <@CIPHER action=hash str="<@CGIPARAM server_name>>. This uses an encrypted form of the domain. 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 Tango 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 123.

For example, the syntax of the assignment to userKey of its default value would be as follows:

```
<@ASSIGN NAME=domainScopeKey VALUE=
<@LITERAL VALUE="<@CIPHER action=hash
str='<@CGIPARAM server_name>>'>
```

Tip It may sometimes be necessary to add security to domainScopeKey, for example, if one copy of Tango is serving application files from several different domains and the system administrator wanted to prevent the administrators of each domain from using the encrypted form of the domain as the scope setting that would give them access to another domain's variables.

You can add a password to the value of domainScopeKey, that is:

```
<@CIPHER action=hash str="<@CGIPARAM
server_name>password">
```

(Substitute your own value for password.)

Because other administrators would not know the password, they could not generate the correct string that is used as `domainScopeKey`.

DSConfig

System scope only

This configuration variable allows you to modify some Tango 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 `Tango 3 Server Preferences` (Mac OS) or `t3server.ini` (Windows and UNIX) file. The contents of the `DSConfig` array are written to and read from a file.

By default, this file is called `Data Source Preferences` (Mac OS) or `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.

For more information on the format and location of this file, see "DSConfigFile" on page 208.

Caution Do not edit the `Data Source Preferences` or `dsConfig.ini` file directly when Tango Server is running. Either stop Tango Server and edit this file, or use the `config.taf` file or your own application files to create or modify the `DSConfig` array, which is then automatically written out to the `Data Source Preferences` or `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`.

The `DSConfig` array has the following structure:

For more information on the structure of the data source configuration file, see "DSConfigFile" on page 208.

Row 0	*type.name*	*type.name*
Row 1 (*maxconnections*)	*n*	*n*
Row 2 (*singlethreaded*)	*1 or 0*	*1 or 0*

The `type` parameter defines the type of data source: ODBC, Oracle or DAM. The `name` parameter defines the name of the data source, the Oracle alias or connect string, or the DAM host name.

The `maxconnections` parameter defines the maximum number of connections that Tango Server makes to the datasource. 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, Tango 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 Tango 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 Tango Server has two open database connections, the next user that tries to connect via Tango 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.

The singlethreaded parameter allows you to override what the data source tells Tango about its thread safety. If you suspect your driver is not thread-safe; that is, cannot be run in a multi-threaded configuration with no ill effects, you can set this parameter to "1" (true), which means that Tango Server only allows one thread to use the driver at any time.

If multiple data sources are using the same driver, which you want to set as singlethreaded, you must specify singlethreaded for each data source.

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 Data Source Preferences (Mac OS) or dsConfig.ini (Windows and UNIX) in the same folder as Tango 3 Server Preferences (Mac OS) or t3server.ini (Windows and UNIX); that is, on Mac OS, the Tango 3 folder within the Preferences folder within the System folder.

The Data Source Preferences file is structured as follows:

```
[Data Sources]
myFirstDS=comment on first ds
mySecondDS=comment on second ds

[myFirstDS]
TYPE=ODBC
MAXCONNECTIONS=0
SINGLETHREADED=1
```

```
[mySecondDS]
TYPE=Oracle
MAXCONNECTIONS=5
SINGLETHREADED=0
```

For more information on the parameters that are set in this file, see "DSConfig" on page 207.

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 the DAM host name.

On server startup, if `DSConfigFile` contains a valid path to an existing file, the contents of the file are used to set up the `DSConfig` variable.

encodeResults

Valid in all scopes

This configuration variable tells Tango whether or not to encode the output sent to the Web browser by Tango in standard HTML format; specifically, changing all high-bit characters to their encoded forms. For example, "é" (a high-bit character, not in the standard ASCII set) is encoded in HTML as `&#233;`. If you would like to send binary data, or are using a character set other than ISO Latin-1, you can set this value to `false`.

The default value of this variable is `true`.

FMDatabaseDir

System scope only

Mac OS only.

The value of this configuration variable is a path telling Tango where to look for local FileMaker Pro databases. When Tango tries to connect to a local data source and finds that the database is not open, it looks in this folder and opens the database (if present).

headerFile

System scope only

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`.

httpHeader

Valid in all scopes

This configuration variable determines the HTTP header used when Tango 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 browser to a different URL.

For more information, see “headerFile” on page 209.

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 Tango Server (the default). To make a permanent change to the HTTP header, use `headerFile`.

The value of `httpHeader scope=local` 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.

itemBufferSize

System scope only

Specifies the size, in bytes, of the largest column value that can be retrieved from a data source. You need to increase this value only if you need to retrieve large values. The default value is `65535` (64K).

license

System scope only

This configuration variable contains your Tango Server registration number. The server runs only if a valid license number is entered.

listenerPort

System scope only

Windows and UNIX only.

This configuration variable sets the port number used by the Tango Server to listen for requests from the Tango 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 `18000`.

lockConfig

System scope only

If this configuration variable is present and set to 'true', the Tango Server configuration file (`Tango 3 Server Preferences` on Mac OS, `t3server.ini` on Windows and UNIX) 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 `Tango 3 Server Preferences`:

```
LOCKCONFIG=true
```

logDir

System scope only

This configuration variable sets the directory used for logging. The log directory should be unique for every Tango Server running on the same machine.

The default value of this variable is `{default path}log.{name of server}`; for example, on Windows NT running the Tango Development Studio, the default value of the variable is `C:\WinNT\Tango3\log.Tango_3_Personal_Server`.

loggingLevel

Valid in all scopes

This configuration variable controls what information is written to the `Tango.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
NoLogging	None
LogLevel1	application file execution, search and post argument values.
LogLevel2	LogLevel1 information plus application file actions.
LogLevel3	LogLevel2 information plus generated SQL, variable and action result values.
LogLevel4	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 `LogLevel3` and `LogLevel4`) only while you need to track down problems with your application files.

The default value of `loggingLevel` is `NoLogging`.

See Also

`debugMode` page 204
`logToResults` page 212

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 Tango 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=LOCAL`.

Note Selecting the Debug mode option in the application file window is equivalent to setting `logToResults scope=LOCAL` to true and `loggingLevelscope=LOCAL` to `LogLevel3`. Because these variables are set with local scope, they are only set to these values for the duration of the file execution.

See Also

`debugMode` page 204
`loggingLevel` page 212

mailDefaultFrom

Valid in all scopes

This configuration variable determines the default `From` value for e-mail messages sent using the Mail action of Tango.

For more details on timed URL processing and startup / shutdown URLs, see the chapter "Using Tango Server" in the *User's Guide*.

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 Tango's timed URL processing, startup / shutdown URLs, and the URL specified in `variableTimeoutTrigger`.

See Also

`mailPort` page 213
`mailServer` page 214

mailPort

System scope only

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 213
mailServer	page 214

mailServer

System scope only

This configuration variable sets the SMTP e-mail server that is used for messages sent with the Mail action.

See Also

mailDefaultFrom	page 213
mailPort	page 213

maxActions

System scope only

If this configuration variable is set to a positive number (the default is zero), the number of Tango 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 217

maxHeapSize

System scope only

UNIX only.

UNIX®

This configuration variable sets the maximum allowable heap size, in bytes. Tango Server restarts itself in a clean state, if its heap size exceeds this value.

The default value is `20000000`.

maxSessions

System scope only

Mac OS only.

This system configuration variable determines the maximum number of sessions Tango 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 Mac OS. 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 Tango Server makes to a particular DAM host.

noSQLEncoding

Valid in all scopes

Tango Enterprise only.

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.

For more information, see "Direct DBMS SQL Auto-Encoding" in the chapter "Using Advanced Database Actions" in the *User's Guide*.

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 10

persistentRestart

System scope only

Windows and UNIX only.

This configuration variable controls how the server handles an automatic restart. An automatic restart is initiated when `maxHeapSize` is exceeded (UNIX only) and when Tango 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`.

See Also

`maxHeapSize` page 214

---

pidFile

System scope only

Windows and UNIX only.

This configuration variable sets the location of a file is used to track the Tango Server process. It should have a unique name for every Tango 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 Tango Server. The characters can be specified by their ASCII number using the `<@CHAR>` tag.

For more information, see “<@CGIPATH>” on page 61.

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 Mac OS 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 222

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.

For more information, see "Arrays" in Chapter 7 of the *User's Guide*.

See Also

`cDelim` page 197

returnDepth

System scope only

This configuration file option sets the maximum number of branch "levels" that you can have in a Tango 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.

The default is 20. Setting this configuration variable to a larger value may increase the memory requirements of Tango 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).

For more information, see "Returning the Value of Arrays" in Chapter 7 of the *User's Guide*.

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 196
aSuffix page 196
cPrefix page 198
cSuffix page 199
rSuffix page 218

---

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).

For more information, see "Returning the Value of Arrays" in Chapter 7 of the *User's Guide*.

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 196
aSuffix page 196
cPrefix page 198
cSuffix page 199
rPrefix page 217

---

shutdownUrl

System scope only

This configuration variable contains the HTTP URL to be requested when Tango Server shuts down.

For more information, see “Startup and Shutdown URL Processing” in the chapter “Using Tango Server” in the *User’s Guide*.

The default value is empty.

See Also

`startStopTimeout` page 219
`startupUrl` page 219

---

startStopTimeout

System scope only

This configuration variable determines how long the Tango Server waits for a response from the URLs that are called when Tango Server shuts down or starts up. The value is specified in seconds.

For more information, see “Startup and Shutdown URL Processing” in Chapter 20 of the *User’s Guide*.

The default value is 60.

See Also

`shutdownUrl` page 218
`startupUrl` page 219

---

startupUrl

System scope only

This configuration variable contains the HTTP URL, if any, that is requested when Tango Server starts up.

For more information, see “Startup and Shutdown URL Processing” in the chapter “Using Tango Server” in the *User’s Guide*.

The default value is empty.

See Also

`shutdownUrl` page 218
`startStopTimeout` page 219

---

staticNumericChars

System scope only

This configuration variable determines when Tango 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 Tango 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. Local 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) data from data sources is automatically stripped of trailing spaces. Possible values are `true` and `false`.

The default value is `true`.

threadPoolSize

System scope only

Windows and UNIX only.

This variable determines the number of worker threads that the Tango Server allocates to process requests. This is the maximum number of requests that Tango 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`.

thousandsChar

Valid in all scopes

(local scope invalid when staticNumericChars=true)

The value of this configuration variable tells Tango 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.

Tango 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 Tango Server is constructing SQL for Search, Insert, Update, and Delete actions. Tango 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 Mac OS, 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 219.

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 local scope have no effect.

See "Understanding Scope" in Chapter 7 of the *User's Guide*.

When `staticNumericChars` has the value `FALSE`, `thousandsChar` works with scope in the standard way.

See Also

`currencyChar`	page 199
`DBDecimalChar`	page 203
`decimalChar`	page 204
`<@DSDATE>`	page 89
`<@DSNUM>`	page 91
`<@DSTIME>`	page 89
`<@DSTIMESTAMP>`	page 89
`staticNumericChars`	page 219

timeFormat

See “dateFormat, timeFormat, timestampFormat” on page 201.

timeoutHTML

System scope only

This configuration variable points to the HTML file that is returned when a query times out in the Tango Server.

The default value of this variable is `timeout.html`.

See Also

`queryTimeout` page 217

timestampFormat

See “dateFormat, timeFormat, timestampFormat” on page 201.

transactionBlocking

System scope only

This configuration variable determines whether Tango Server blocks other processes during a transaction. Transactions begin with the Begin Transaction action and end when an End Transaction action is reached, an error occurs (automatic rollback), or the end of an application file or a Return action is reached (automatic commit).

For more information, see “Creating Database Transactions” in the *User’s Guide*.

This variable accepts TRUE or FALSE as values. The default value is TRUE. It is read only at startup; a restart is required to effect changes to it.

Caution Setting this configuration variable to FALSE may cause poor performance due to record contention when multiple users are executing transactions with Tango.

userAgent

Valid in all scopes

The value of this configuration variable is used in the header of HTTP requests sent by Tango 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. (Tango 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 page, including browser-specific HTML for additional features, when the browser is Netscape Navigator or Internet Explorer.

The default value of `userAgent` is empty, causing Tango Server URL requests to use `"Tango Application Server/[version]([platform])"`, where `[version]` and `[platform]` are the values returned by `<@VERSION>` and `<@PLATFORM>` meta tags.

See Also

`<@URL>` page 178

userKey, altuserKey

Local, domain and system scopes

These variables set the key used to identify users in Tango.

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, Tango 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. Tango gives you several options for specifying how Tango 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 Tango 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.

When you assign a value to userKey and altUserKey, you must tell Tango 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 178.

The syntax of the assignment to userKey of its default value would be as follows:

```
<@ASSIGN NAME=userKey VALUE=<@LITERAL
VALUE="<@USERREFERENCE>">
```

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 10.

For example, <@VAR NAME=userKey> might return <@USERREFERENCE>, indicating that user configuration variables are keyed on the Tango user reference ID assigned to each user. 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 <@USERREFERENCE>. The default value of altUserKey is empty.

See Also

<@CGIPARAM> page 58
<@USERREFERENCE> page 181
<@VAR> page 184

---

validHosts

System scope only

Windows and UNIX only.

UNIX®

This configuration variable specifies a list of hosts from which Tango Server accepts Tango 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 Tango Server.

varCachePath

System scope only

This specifies a folder to which Tango writes all variables when it is shutdown, and re-reads those variables from when Tango is started.

Note Variables continue to expire in the usual fashion; if you restart Tango 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 Tango Server}`; for example, on Windows NT when running the Tango Development Studio, the value of the variable is `C:\WinNT\Tango3\variables.Tango_3_Personal_Server`.

variableTimeout

User, domain, 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 Tango Server. For domain variables, the expiry timer is reset each time a user from the domain accesses Tango Server.

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.

See Also

`variableTimeoutTrigger` page 226

---

variableTimeoutTrigger

User and domain scopes

Just before a user's or a domain'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.

The URL in this configuration variable cannot contain meta tags because the trigger mechanism does not evaluate meta tags. Nevertheless, you can include user- 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 213
`userAgent` page 223
`variableTimeout` page 225

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V