TalentSoft™

Web+™ 2.1

Language Reference

Copyright Notice

TalentSoft welcomes any comments and suggestions regarding this document and the Web+™ software.

TalentSoft™ / Talent Information Management, LLC.
Address: 900 Nicollet Mall, Ste. 700
Minneapolis, MN 55402, USA
Tel: (612) 338-8883
Fax: (612) 904-0010
Email: info@talentsoft.com
URL: http://www.talentsoft.com

Contents

Introduction

Terms Used in This Manual

argumentlist [*expression* | *parameter* := *expression*][, ...]

A list of zero or more *expressions* that are assigned to the *parameters* of the subroutine, function or property.

- A positional *parameter* may be skipped by omitting the *expression*. Only optional *parameters* may be skipped.
- Positional *parameter* assignment is done with *expression*. Each *parameter* is assigned in turn. By name *parameter* assignment may follow.
- By name *parameter* assignment is done with *parameter* := *expression*. All following *parameters* must be assigned by name.

Conditional expression

An expression that returns a numeric result. If the result is zero then the conditional is **False**. If the result is non-zero then the conditional is **True**.

Examples:

```
0                <!-- False -->
-1               <!-- True -->
#(X gt 20)#      <!-- True if X is greater than 20 -->
#S = "hello"#    <!-- True if S equals "hello" -->
```

expression An *expression* that returns the appropriate result.

statement A single command.

Examples:

```
<webBeep>
```

name An identifier that names a variable or a user defined subroutine, function or property. Identifiers start with a letter. Following characters may be a letter, an underscore, or a digit.

Examples:

```
Count
DaysTill2000
Get_Data
```

number An expression that returns a numeric result.

Examples:

```
10236
3.14159
1.2E12
Count
Count - 1
InStr(S, "A")
```

parameter [[Optional] [| ByVal | ByRef] | *parameterarray*] *parameter*[( )] [As type]

The *parameter* receives the value of the associated expression in the subroutine, function or property call. (See arglist.)

- An Optional *parameter* may be omitted from the call. It must be a Variant type. All parameters following an Optional parameter must also be Optional.
- p*arameterarray* may be used on the final *parameter*. It must be an array of Variant type. It must not follow any Optional parameters. The *parameterarray* receives all the expressions at the end of the call as an array. If LBound(*parameter*) > UBound(*parameter*) then the *parameterarray* didn't receive any *expressions*.
- If the *parameter* is not ByVal and the expression is merely a variable then the *parameter* is a reference to that variable (ByRef). (Changing *parameter* changes the variable.) Otherwise, the *parameter* variable is local to the subroutine, function or property, so changing its value does not affect the caller.
- Use *parameter*( ) to specify an array *parameter*. An array *parameter* must be referenced and can not be passed by value. The bounds of the *parameter* array are available via LBound( ) and UBound( ).
- Property Get, Let and Set blocks do not allow Optional or *parameterarray* parameter types.

property An object provides methods and properties. Properties may be used as values (like a function call) or changed (using assignment syntax).

If the property name contains characters that are not legal in a name, surround the property name with [ ].

Example:

```
Cookie.[Last Visited]
```

statement Zero or more statements. A statement is at least one script line long. Do, For, If (multiline), Select Case, While, and With statements are always more than one line long. A single line statement continues on the next line if it ends a line with a space and an underscore ' _'

```
MyStr = "This long string is easier to read, " + _
      "if it is broken across two lines."
Debug.Print MyStr
```

string An expression that returns a string result.
Examples:
```
"Hello"
MyStr
MyStr & " Goodbye"
Mid(MyStr, 2)
```

stringvariable A variable that holds one string value.

Example:
FirstName

variable A variable holds either a string, a numeric value or an array of values depending on its type.

EXAMPLE	DESCRIPTION
WEBP.EXE TEST.WML	UPPERCASE letters indicate filenames, registers, and terms used at the operating system command level.
`<webSet #I# = #Rnd(2)# >`	**Bold** type indicates Web+™ constants, functions, keywords, operators, language-specific characters. Within discussions of syntax, bold type indicates that the text must be entered exactly as shown.
Expression	Words in *italics* indicate placeholders for information you must supply, such as a filename. Italic type is also used occasionally for emphasis in the text.
[option]	Items inside square brackets are optional.
Mode="{**Read**|**Write**|**Append**}"	Braces and a vertical bar indicate a choice among two or more items. You must choose one of these items unless square brackets ([ ]) surround the braces.
`<webSet #I# = 1 >`	Fixed spacing font (Courier New) is used for examples, user input, program output, and error messages in text.
CL [option...] file...	Three dots (an ellipsis) following an item indicate that more items having the same form may appear.

While()
{…
}
	A column or row of three dots tells you that part of an example program has been intentionally omitted.

CTRL+ENTER	Small capital letters are used to indicate the names of keys on the keyboard. When you see a plus sign (+) between two key names, you should hold down the first key while pressing the second. The carriage-return key, sometimes marked as a bent arrow on the keyboard, is called ENTER.
"argument"	Quotation marks enclose a new term the first time it is defined in text.
"C string"	Some C constructs, such as strings, require quotation marks.
Variable	A variable holds either a string, a numeric value or an array of values.
Hyper Text Markup Language (HTML)	The first time an acronym is used, it is usually spelled out.
See Using Projects	Hypertext cross-references are in green underlined text.
CEnterDlg;	The blue triangle adjacent to the blue code indicates that it has been altered from a previous example, usually because you are being instructed to edit it.

Language Reference

Data Types

Number Type

Description Number type includes: integer, decimal, real number.

Syntax <**webSet** #V*ariableName# = number*>

Remarks In any math computation, Web+ only keeps 12 decimal places.

See Also

Example

```
<!-- The following are equivalent, quotes are optional -->
<webSet #A# = 1.23 >
<webSet #A# = "1.23" >

<!-- The following shows simple math processing -->
<webSet #B# = # 1 + 0.12345678901234567890 # >
<!-- Result:  #B# = 1.123456789012  (because Web+ only retains
12 decimal places) -->
```

String Type

Description Web+ strings are variable-length, which can contain up to approximately 16,000 characters (16 K) long. Sting type values must be enclused by double-quotes (").

Syntax <**webSet** #V*ariableName# = string*>

Remarks Size limit, 16 K.

See Also

Example

```
<!-- The following are equivalent, quotes are optional -->
<webSet #A# = "TalentSoft Web+" >
<webSet #A# = "1.23" >

<!-- The following shows simple math processing -->
<webSet #B# = # 1 + 0.12345678901234567890 # >
<!-- Result:  #B# = 1.123456789012  (because Web+ only retains
12 decimal places) -->
```

Array Type

Description An array is a set of sequentially indexed elements having the same type. Each element of an array has a unique index number that identifies it. Changes made to an element of an array do not affect the other elements..

Syntax Assigning a single value to an array index:

```
<webSet #variable[index]# = expression>
```

Assigning a list of values directly to an array:

```
<webSet #variable[]# = "d value0 d value1 d value2 d value3"
```

where `d` represents the delimiter character.

Remarks

See Also **ArrayCount** function

Example

```
<!-- The following are equivalent, quotes are optional -->
<webSet #A[]# = ",0,1,2,3,,5,6,,,20,30">
<webFor #i# = 0 to 10>
    <webSet #B[i]# = #Int (Rnd (2) * 100)#>
</webFor>
<webFor #i# = 0 to 10>
    <webPrint>(#A[n]#, #B[n]#);</webPRINT>
</webFor>
```

Web+ Output:

```
(0, 77); (1, 95); (2, 85); (3, 38); (, 10); (5, 100); (6, 53);
(, 79); (, 33); (20, 76); (30, 1);
```

Variable Assignment and Output Statements

webSet Statement

Description Assigns the value of a constant or an expression to a variable.

Syntax <**webSet** #V*ariableName# = expression*>

Remarks None.

See Also

Example

```
<webSet #I# = 1 >
<webSet #I# = #(I + 1)# >
<webSet #PathName# = "C:\temp\" >
<webSet #I# = #Rnd(2)# >
<webSet #Total# = #(100 * 200 + 300 / 400)# >


<!-- You can use <webSet> to return the result of the comparison
between two strings -->
<webSet #string1# = "webplus"><webSet #string2# = "webplus">
<webSet #samestring# = #string1 = string2#>
<webPrint>String Comparison Result = #samestring#</webPrint>
Web+Output:
    String Comparison Result = 1
```

webPrint Statement

Description Prints information based on the records returned by a query.

Syntax <**webPrint**> V*ariableName, Expressions* </**webPrint**>

Remarks

See Also **webPrint, webPrintTable in database section**

Example

```
<webPrint>
#StateID#  #StateName#
</webPrint>
```

Math and Logical Functions

webEval Statement

Description Evaluates an expression and returns a value.

Syntax

```
<webEval CodeVar=Variable Ret=Variable>
<webEval CodeFile=File Name=Exp Ret=Variable>
```

Remarks None.

See Also

Example

```
<webSet #A# = "##(1+2)##">
<webEval CodeVar=#A# Ret=#Z#>
<webPrint>A=#A# Z=#Z#  </webPrint>
```

Web+ Output:

A= #(1+2)# Z=3

ArrayCount Function

Description Returns the number of items in an array.

Syntax

```
#ArrayCount(name of the array)#
```

Remarks

See Also

Example

```
<!-- This example illustrates the number of items in an array --
>
<webSet #A[]# = ",1,3,5,7,9,,13,,,19">
<webFor #i# = 0 to 10>
   <webSet #B[i]# = #Int(Rnd (2) * 100)#>
</webFor>
<webPrint>Nbr of items in array A = #ArrayCount(A[])#
          Nbr of items in array B = #ArrayCount(B)# </webPrint>
Web+ Output:
Nbr of items in array A = 10
Nbr of items in array B = 11
```

Abs Function

Description Returns the absolute value of a number..

Syntax #**Abs**(*number*)#

Argument	Required	Description
number	Yes	The argument number can be any valid numeric expression. The absolute value of a number is its unsigned magnitude. For example, ABS(-1) and ABS(1) both return 1.

Remarks

See Also

Example

```
<!-- This example illustrates how to calculate the difference
between two numbers, X and Y -->
<webSet #Difference# = #Abs(X - Y)#>
```

Int Function

Description Return the integer portion of a number.

Syntax #**Int**(*number*)#

Argument	Required	Description
number	Yes	The argument number can be any valid numeric expression. Int removes the fractional part of number and return the resulting integer value. For example, Int(1.5) returns 1 and Int(-1.5) returns -1.

Remarks

See Also

Example

```
<!-- This example illustrates how to calculate the difference
between two numbers, X and Y -->
<webSet #Difference# = #Int(X - Y)#>
```

IncrementValue Function

Description Increments the value of the numeric expression by 1.

Syntax #IncrementValue(*number*)#

Argument	Required	Description
number	Yes	Numeric expression to be incremented by 1.

Remarks

See Also **DecrementValue**, **+** Operator

Example

```
<webPrint>
The next available ID is:  #IncrementValue(UserID)#
</webPrint>
```

DecrementValue Function

Description Decrements the value of the numeric expression by 1.

Syntax #DecrementValue(*number*)#

Argument	Required	Description
number	Yes	Numeric expression to be decreased by 1.

Remarks

See Also **IncrementValue**, **-** Operator

Example

```
<webPrint>
The previous User ID is:  #DecrementValue(UserID)#
</webPrint>
```

Rnd Function

Description Returns a random number greater than or equal to zero and less than one.

Syntax #**Rnd**(*n*)#

Argument	Required	Description
n	Yes	Integer expression indicating the precision (number of digits/decimal places) to return. The integer must be between 1 and 12. If *n* is greater than 12, the function only returns the first 12 digits and the rest are truncated.

Remarks

See Also

Example

```
<!-- This example illustrates how to generate a random integer
number between 0 and 10 -->
<webSet #I# = #Rnd(1) * 10 #>
```

DateDiff Function

Description Returns a number containing the number of time intervals between two specified dates.

Syntax

```
#DateDiff(interval, startdate, enddate)#
The DateDiff function uses the following parts:
```

Argument	Required	Description
interval	Yes	String expression that is the interval of time you use to calculate the difference between startdate and enddate.
startdate	Yes	The starting date for calculation
enddate	Yes	The ending date for calculation

You can use the DateDiff function to determine how many specified time intervals exist between two dates. For example, you might use DateDiff to calculate the number of days between two dates or the number of hours between now and the last update time.

The following table lists the valid time periods and their interval values.

These intervals are also used by the Format function.

Interval Label	Available	Time Period Description
yyyy	No *	Year
q	No *	Quarter
m	No *	Month
y	No *	Day of year
d	Yes	Day
w	No *	Weekday
ww	No *	Week
h	Yes	Hour
n	Yes	Minute
s	Yes	Second

* No means not available in Web+(TM) version 2.1 but will be available in future versions of Web+ .

If you want to know the number of days between startdate and enddate, you can use either Day or Hour ("d" or "h").
When interval is Weekday ("w"), DateDiff returns the number of weeks between the two dates. If startdate falls on a Monday, DateDiff counts the number of Mondays until enddate. It counts startdate but not enddate. If interval is Week ("ww"), however, the DateDiff function returns the number of calendar weeks between the two dates. It counts the number of Sundays between startdate and enddate. DateDiff counts startdate if it falls on a Sunday; but it doesn't count enddate, even if it does fall on a Sunday.

If startdate refers to a later point in time than enddate, the DateDiff function returns a negative number.

Remarks

See Also

Example

```
<!-- This example illustrates the day difference between the
current date time and 1996-06-06 13:02:54 -->
#DateDiff("d", CurrentDateTime(),1996-06-06 13:02:54)#>
```

WeekDay Function

Description Returns an integer between 1 (Sunday) and 7 (Saturday) that represents the day of the week for a date argument.

Syntax #Weekday(date*expression*)#

Argument	Required	Description
dateexpression	Yes	The dateexpression can be any string expression that can represent a date and/or time from January 1, 1970 through Decemeber 31, 9999, where January 1, 1900 is 2. If dateexpression is Null, Weekday function will return a Null.

Remarks

See Also **TimeFormat, Format**

Example

```
<webPrint>#WeekDay("January 1, 1990")#</webPrint>
```

IsDate Function

Description Returns the value 1 when the expression can be converted to a date, otherwise, returns 0.

Syntax #IsDate(*expression*)#

Argument	Required	Description
dateexpression	Yes	*expression* can be any string expression that can represent a date and/or time from January 1, 1970 through December 31, 9999. If *expression* is Null, IsDate will return a Null.

Remarks

See Also **TimeFormat, WeekDay, CurrentDate()**

Example

```
<webPrint>#IsDate(CurrentDate())#</webPrint>
Web+Output:  1
```

IsNumeric Function

Description Returns the value 1 when the expression can be converted to a numeric data type, otherwise, returns 0.

Syntax #IsNumeric(expression)#

Argument	Required	Description
expression	Yes	The expression can be any string expression or numeric constant.

Remarks

See Also

Example

```
<webPrint>#IsNumeric(3)#</webPrint>
Web+Output:  1
```

UploadFile Function

Description Get the logical file name and upload the file on the Web+™ server's machine.

Syntax #**UploadFile(userfile)**#

Remarks

Example

```
<!-- Upload the file to the server -->
<webIf #Parameterexists(Done)#>
    Uploading File Name:
    <webprint>#UploadFile(userfile)#</webprint>
    <webAbort>
</webIf>
<FORM ENCTYPE="multipart/form-data" METHOD="POST"
ACTION="upload.wml">
     <INPUT TYPE=FILE NAME=userfile>
     <INPUT TYPE="Submit" NAME="Done" VALUE="Submit">
</FORM>
```

Environmental Information Functions

CurrentPath Function

Description Returns the current path of the script.

Syntax `#CurrentPath()#`

Remarks

See Also `./`

Example
```
<webPrint> The current script path is: #CurrentPath()#
</webPrint>
```

GetTempFileName Function

Description Returns the system level temp file name to be used for file i/o.

Syntax `#GetTempFileName()#`

Remarks Returns a temporary file name with logical path \temp\. So you have to predefine a Logical /temp/ path in the webpsvr.ini and a physical path. Please note, if the NT's environment variable TMP is set, then the temporary will be under the TMP's directory no matter what physical path you defined in the webpsvr.ini, and it will be in the physical path you defined if TMP is undefined.

path.

See Also

Example
```
<webPrint> The temporary file name is: #GetTempFileName()#
</webPrint>
```

CurrentDate Function

Description Returns the current system date of the Web+™ server.

Syntax `#CurrentDate()#`

Remarks

See Also `CurrentTime(), CurrentDateTime()`

Example <webPrint> Today's date is: #CurrentDate()# </webPrint>

Web+Output:

```
Today's date is: 1996-06-06
```

CurrentTime Function

Description Returns the current system Time of the Web+™ server.

Syntax #CurrentTime()#

Remarks

See Also CurrentDate(), CurrentDateTime()

Example <webPrint> The time now is: #CurrentTime()# </webPrint>

Web+Output:

```
The time now is: 13:02:54
```

CurrentDateTime Function

Description Returns the current system date and time of the Web+™ server.

Syntax #CurrentDateTime()#

Remarks

See Also CurrentDate(), CurrentTime()

Example <webPrint> Now is: #CurrentDateTime()# </webPrint>

Web+ Output

```
Now is: 1996-06-06 13:02:54
```

CGI Environment Variables

The following description of available CGI Environment Variables is taken from the NCSA CGI Specification (http://hoohoo.ncsa.uiuc.edu/cgi/interface.html).

In order to pass data about the information request from the server to the script, the server uses command line

arguments as well as environment variables. These environment variables are set when the server executes the gateway program.

Variable	Explanation
SERVER_SOFTWARE	The name and version of the information server software answering the request (and running the gateway). Format: name/version.
SERVER_NAME	The server's hostname, DNS alias, or IP address as it would appear in self-

	referencing URLs.
GATEWAY_INTERFACE	The revision of the CGI specification to which this server complies. Format: CGI/revision.
SERVER_PROTOCOL	The name and revision of the information protocol this request came in with. Format: protocol/revision.
SERVER_PORT	The port number to which the request was sent.
REQUEST_METHOD	The method with which the request was made. For HTTP, this "GET", "HEAD", "POST", etc.
PATH_INFO	The extra path information, as given by the client. In other words, scripts can be accessed by their virtual pathname, followed by extra information at the end of this path. The extra information is sent as PATH_INFO.
PATH_TRANSLATED	The server provides a translated version of PATH_INFO, which takes the path and does any virtual-to-physical mapping to it.
SCRIPT_NAME	A virtual path to the script being executed, used for self-referencing URLs.
QUERY_STRING	The information which follows the ? in the URL that referenced this script. This is the query information.
REMOTE_HOST	The hostname making the request. If the server does not have this information, it sets REMOTE_ADDR and leaves this unset.
REMOTE_ADDR	The IP address of the remote host making the request.
AUTH_TYPE	If the server supports user authentication, and the script is protected, this is the protocol-specific authentication method used to validate the user.
REMOTE_USER	If the server supports user authentication, and the script is protected, this is the username they have authenticated as.
REMOTE_IDENT	If the HTTP server supports RFC 931 identification, then this variable will be set to the remote user name retrieved from the server. Usage of this variable should be limited to logging only.
CONTENT_TYPE	For queries which have attached information, such as HTTP POST and PUT, this is the content type of the data.
CONTENT_LENGTH	The length of the said content as given by the client.

In addition to these, the header lines received from the client, if any, are placed into the environment with the prefix HTTP_ followed by the header name. Any - characters in the header name are changed to _ characters. The server may exclude any headers which it has already processed, such as Authorization, Content-type, and Content-length. If necessary, the server may choose to exclude any or all of these headers if including them would exceed any system environment limits.

An example of this is the HTTP_ACCEPT variable which was defined in CGI/1.0. Another example is the header User-Agent.

Variable	**Explanation**
HTTP_REFERER	The referring document (document which linked to or submitted form data to this one).
HTTP_USER_AGENT	The browser the client is using to send the request. Format: software/version library/version.

Web+™ Server Configuration Functions

webIniGet Function

Description Returns the current value (stored in memory by Web+™ server) of the specified Web+™ configuration key name.

Syntax #webIniGet(*webpluskeyname,*[*AdminPassword*])#

webpsvr.ini Key Name	Configurable	Description
AdminPassword	No	Web+ administration password. Remarks This admin password is optional. If you set the AdminIniPasswordRequired under the [Admin] Section to 0, i.e., it does not require any password to process

Remarks

See Also

Example

```
<webPrint>
Your Web+TM LicenseID is: #webIniGet("LicenseID", "Passwd")#
The Maximum Number of Concurrent Users allowed by your Web+
License is: #webIniGet("LicMaxConcurrentUsers", "Passwd")#
Your Web+ Administrator's Email is: #webIniGet("AdminEmail",
"Passwd")#
</webPrint>
```

webIniPut Function

Description Put the specified Web+™ value into the webpsvc.ini configuration file.

Syntax

```
<webIniPut SectionName, KeyName, Value,
[Password=AdminPassword]>
```

webpsvr.ini Key Name	Configurable	Description
AdminPassword	No	Web+ administration password. Remarks This admin password is optional. If you set the AdminIniPasswordRequired under the [Admin] Section to 0, i.e., it does not require any password to process

SectionName	Yes	The Section Name in the webpsvr.ini.
Keyname	Yes	The Key Name in the webpsvr.ini

Remarks

See Also

Example

webIniRefresh Function

Description Refreshes the values of Web+™ configuration key names.

Syntax <**webIniRefresh**, [Password=*AdminPassword*]>

Remarks

See Also

Example <**webIniRefresh**, [**Password="abc"**]>

String Processing Functions

Asc Function

Description Returns the ASCII value for the first character in a string.

Syntax #**Asc**(*string*)#

The *string* parameter is any valid string expression. If the *string* contains no characters, a run-time error occurs.

See Also **Appendix A - ANSI Character Set.**

Example

```
<webPrint>
The ASCII value of "A" is #Asc("A")#.<BR>
The ASCII value of "a" is #Asc("a")#.<BR>
The ASCII value of "ABC" is #Asc("ABC")#.<BR>
</webPrint>
```

Chr Function

Description Returns a one character string for the ASCII value.

Syntax #**Chr**(*number*)#

The *number* parameter must be a valid numeric expression.

Remarks Numbers from 0 to 31 are nonprintable ASCII values.

See Also

Example

```
<webPrint>
The ASCII character for 65 is #Chr(65)#.<BR>
</webPrint>
```

LCase Function

Description Returns the given String value as an all lowercase String value.

Syntax #**LCase**(*string*)#

Returns the string value of *string* after all characters have been converted to lowercase.

Remarks

See Also **UCase Function**

Example

```
<webPrint>
The word "TalentSoft" converted to lower case is
#LCase("TalentSoft")#.
</webPrint>
```

UCase Function

Description Returns the given String value as an all uppercase String value.

Syntax `#UCase(string)#`

Returns the string value of *string* after all characters have been converted to uppercase.

Remarks

See Also **LCase**

Example

```
<webPrint>
The word "TalentSoft" converted to upper case is
#UCase("TalentSoft")#.
</webPrint>
```

LTrim Function

Description Removes leading spaces (spaces on the left side) of a string

Syntax `#LTrim(string)#`

Remarks

See Also **RTrim, Trim**

Example

```
<webSet #A# = "     TalentSoft     " >
<webPrint>
After removing the leading spaces from the word #A#, the result
is #LTrim(A)#.
</webPrint>
```

RTrim Function

Description Removes the trailing spaces (spaces on the right side) of a string

Syntax `#RTrim(string)#`

Remarks

See Also **LTrim, Trim**

Example

```
<webSet #A# = "     TalentSoft     " >
<webPrint>
After removing the trailing spaces from the word #A#, the result
is #RTrim(A)#.
</webPrint>
```

Trim Function

Description Removes the leading and trailing spaces (spaces on both sides) of a string

Syntax #**Trim**(*string*)#

Remarks

See Also **RTrim, LTrim**

Example
```
<webSet #A# = "     TalentSoft     " >
<webPrint>
After removing the leading and trailing spaces from the word
#A#, the result is #Trim(A)#.
</webPrint>
```

Len Function

Description Returns the number of characters in a String value or the number of bytes needed to store the value in a variable.

Syntax #**Len**(*string*)#

Returns the number of characters in the string value *s*.

Remarks

See Also **InStr Function**, Mid Function.

Example
```
<webPrint>
The number of characters in the word "TalentSoft" is
#Len("TalentSoft")#.
</webPrint>
```

Left Function

Description Returns the leftmost *n* characters of a string argument.

Syntax #Left(*strexpr*, *n*)#

Argument	Required	Description
strexpr	Yes	String expression from which the leftmost characters are returned. This can be any string expression.
n	Yes	Integer expression indicating how many characters to be returned. It must be between 0 and approximately 65,535, inclusive. If *n* is 0, the return value is a zero-length string. If *n* is greater than or equal to the number of characters in *strexpr*, the entire string is returned.

Remarks To find the number of characters in *strexpr*, use **Len**(*strexpr*).

See Also **InStr Function**, **Right Function, Mid Function.**

Example
```
<webPrint>
The left 5 characters from the word "TalentSoft" are
#Left("TalentSoft", 5)#.
</webPrint>
```

Right Function

Description Returns the rightmost *n* characters of a string argument.

Syntax #Right(*strexpr*, *n*)#

Argument	Required	Description
strexpr	Yes	String expression from which the leftmost characters are returned. This can be any string expression.
n	Yes	Integer expression indicating how many characters to return. It must be between 0 and approximately 65,535, inclusive. If *n* is 0, the return value is a zero-length string. If *n* is greater than or equal to the number of characters in *strexpr*, the entire string is returned.

Remarks To find the number of characters in *strexpr*, use **Len**(*strexpr*).

See Also **InStr Function**, **Left Function, Mid Function.**

Example
```
<webPrint>
The right 4 characters from the word "TalentSoft" are
#Right("TalentSoft", 4)#.
</webPrint>
```

Mid Function

Description Returns the requested number of characters from a given starting position of a String expression.

Syntax #Mid(*stringexpr*, *start* [, *length*])#

Argument	Required	Description
stringexpr	Yes	String expression from which another string is created. This can be any string expression.
start	Yes	Integer expression that indicates the character position in *stringexpr* at which the part to be taken begins.
length	No	Integer expression indicating the number of characters to return.

Remarks The arguments *start* and *length* must be between 1 and approximately 65,535, inclusive. If length is omitted or if there are fewer than length characters in the text (including the character at start), all characters from the start position to the end of the string are returned.
If *start* is greater than the number of characters in *stringexpr*, a zero-length

string is returned.
Use the **Len function** to determine the length of the *stringexpr* parameter.

See Also **InStr Function**, **Len Function**

Example

```
<webSet #MyString# = "Web+ is Great">
<webPrint>
My string variable contains the value "#MyString#".<BR>
The first word in my string is "#Mid(MyString, 1, 4)#".<BR>
The second word in my string is "#Mid(MyString, 6, 2)#".<BR>
The third word in my string is "#Mid(MyString, 9)#".<BR>
</webPrint>
```

InStr Function

Description Returns the position of the first occurrence of one string within another string.

Syntax #InStr([*start*], *stringexpr1*, *stringexpr2*, [*mode*])#

Argument	Required	Description
start	No	Integer expression that sets the starting position for each search; start must be between 1 and approximately 65,535. If *start* is omitted, the search of strexpr1 begins at the first character position.
Stringexpr1	Yes	String expression being searched.
Stringexpr2	Yes	String expression being sought.
Mode	No	If *mode* is 0, string comparison is case-sensitive; so, for example, "T" doesn't match "t". If *mode* is 1, string comparison is not case-sensitive; so, for example, "T" matches "t". If *mode* is omitted, the default mode is 0.

Remarks If *stringexpr2* is found within *stringexpr1*, InStr returns the position at which the match was found. If *stringexpr2* is zero-length, start is returned. If start is greater than *stringexpr1*, *stringexpr1* is zero-length, or *stringexpr2* can't be found, InStr returns 0.

See Also **Mid Function.**

Example

```
<webPrint>
The position of the first letter "e" in "TalentSoft Web+" is
#InStr("TalentSoft Web+", "e")#.
</webPrint>
```

Web+ Output:

```
The position of the first letter "e" in "TalentSoft Web+" is 4
```

StrReplace Function

Description Replace the string expression 1 to string expression 2.

Syntax #StrReplace(*stringexpr*, *stringexpr1*, *stringexpr2*, [*mode*])#

Argument	Required	Description
stringexpr	Yes	String expression being searched.
Stringexpr1	Yes	String expression being replaced by *stringexpr2*.
Stringexpr2	Yes	String expression to replace *stringexpr1*.
Mode	No	If *mode* is 0, string comparison is case-sensitive; so, for example, "T" doesn't match "t". If *mode* is 1, string comparison is not case-sensitive; so, for example, "T" matches "t". If *mode* is omitted, the default mode is 0.

Remarks

See Also

Example

```
<webPrint>
I will now replace the string "+" in "TalentSoft Web+"
#StrReplace("TalentSoft Web+", "+", "Plus")#.
</webPrint>
```

Format Function

Description Formats a number, date, time, or string according to instructions contained in a format expression.

Syntax #Format(*expression*, *format*)#

Argument	Required	Description
expression	Yes	Numeric or string expression to be formatted.
format	Yes	Format expression a string of display-format characters that specify how the expression is to be displayed or the name of a commonly used format that has been predefined in Web+™.

To format numbers, you can use the commonly used formats that have been predefined in Web+™ or you can create user-defined formats with standard characters that have special meaning when used in a format expression.
The following table shows the predefined numeric format names you can use and the meaning of each:

Format Name	Meaning
General Number	Display the number as is, with no thousand separators.
Currency	Display number with thousand separator, if appropriate; display negative numbers enclosed in parentheses; display two digits to the right of the decimal separator and also with the ($) dollar sign.
Fixed	Display at least one digit to the left and two digits to the right of the decimal separator.
Standard	Display number with thousand separator, if appropriate; display two digits to the right of the decimal separator.

Percent	Display number multiplied by 100 with a percent sign (%) appended to the right; display two digits to the right of the decimal separator.
Scientific	Use standard scientific notation.
Yes/No	Display No if number is 0, otherwise display Yes.
True/False	Display False if number is 0, otherwise display True.
On/Off	Display Off if number is 0, otherwise display On.

The following table shows the characters you can use to create user-defined number formats and the meaning of each:

Format Character	Meaning
0	Digit placeholder. Display a digit or a zero. If there is a digit in the expression being formatted in the position where the 0 appears in the format string, display it; otherwise, display a zero in that position. If the number being formatted has fewer digits than there are zeros (on either side of the decimal) in the format expression, leading or trailing zeros are displayed. If the number has more digits to the right of the decimal separator than there are zeros to the right of the decimal separator in the format expression, the number is rounded to as many decimal places as there are zeros. If the number has more digits to the left of the decimal separator than there are zeros to the left of the decimal separator in the format expression, the extra digits are displayed without modification.
?	Digit placeholder. Display a digit or nothing. If there is a digit in the expression being formatted in the position where the ? appears in the format string, display it; otherwise, display nothing in that position. This symbol works like the 0 digit placeholder, except that leading and trailing zeros aren't displayed if the number has the same or fewer digits than there are ? characters on either side of the decimal separator in the format expression.
.	Decimal placeholder. The decimal placeholder determines how many digits are displayed to the left and right of the decimal separator. If the format expression contains only number signs to the left of this symbol, numbers smaller than 1 begin with a decimal separator. If you want a leading zero to always be displayed with fractional numbers, use 0 as the first digit placeholder to the left of the decimal separator instead.
%	Percentage placeholder. The expression is multiplied by 100. The percent character (%) is inserted in the position where it appears in the format string.
,	Thousand separator. The thousand separator separates thousands from hundreds within a number that has four or more places to the left of the decimal separator. Standard use of the thousand separator is specified if the format contains a comma surrounded by digit placeholders (0 or ?). Two adjacent commas or a comma immediately to the left of the decimal separator (whether or not a decimal is specified) means "scale the number by dividing it by 1000, rounding as needed." You can scale large numbers using this technique. For example, you can use the format string "??0,," to represent 100 million as 100. Numbers smaller than 1 million are displayed as 0. Two adjacent commas in any position other than immediately to the left of the decimal separator are treated simply as specifying the use of a thousand separator.

Remarks

See Also **DateFormat, TimeFormat, NumberFormat, DecimalFormat, YesNoFormat.**

Example

```
<webSet #N1# = 12345.6789>
<webPrint>
Format(N1, "General Number") = #Format(N1, "General Number")#
Format(N1, "Currency")       = #Format(N1, "Currency")#
Format(N1, "Fixed")          = #Format(N1, "Fixed")#
Format(N1, "Standard")       = #Format(N1, "Standard")#
Format(N1, "Percent")        = #Format(N1, "Percent")#
Format(N1, "Discard Decimal")= #Format(N1, "Discard Decimal")#
Format(N1, "Scientific")     = #Format(N1, "Scientific")#
Format(N1, "Yes/No")         = #Format(N1, "Yes/No")#
Format(N1, "True/False")     = #Format(N1, "True/False")#
Format(N1, "On/Off")         = #Format(N1, "On/Off")#
Format(N1, "0.00")           = #Format(N1, "0.00")#
Format(N1, "?,??0.00")       = #Format(N1, "?,??0.00")#
Format(N1, "0")              = #Format(N1, "0")#
Format(N1, "$?,??0.00")      = #Format(N1, "$?,??0.00")#
Format(N1, "0.00%")          = #Format(N1, "0.00%")#
Format(N1, "0%")             = #Format(N1, "0%")#
Format(N1, "A000,000.00")    = #Format(N1, "A000,000.00")#
Format(N1, "A000,000.00-BB") = #Format(N1, "A000,000.00-BB")#
Format(N1, "A000,000.00-BB") = #Format(N1, "A000,000.00-BB")#
Format(N1, "A000,000.00000") = #Format(N1, "A000,000.00000")#
</webPrint>
```

Example Output

```
Format(N1, "General Number") = 12345.6789
Format(N1, "Currency")       = $12,345.68
Format(N1, "Fixed")          = 12345.68
Format(N1, "Standard")       = 12,346.68
Format(N1, "Percent")        = 1234567.89%
Format(N1, "Discard Decimal")= 12345
Format(N1, "Scientific")     = Scientific
Format(N1, "Yes/No")         = Yes
Format(N1, "True/False")     = True
Format(N1, "On/Off")         = On
Format(N1, "0.00")           = 12345.68
Format(N1, "?,??0.00")       = 12,345.68
Format(N1, "0")              = 12346
Format(N1, "$?,??0.00")      = $12,345.68
Format(N1, "0.00%")          = 1234567.89%
Format(N1, "0%")             = 1234568%
Format(N1, "A000,000.00")    = A012,345.68
Format(N1, "A000,000.00-BB") = A012,345.67-BB
Format(N1, "A000,000.00-BB") = A012,345.67-BB
Format(N1, "A000,000.00000") = A012,345.67890
```

DateFormat Function

Description Displays a database date/time or date field using the DD-MMM-YY format. 12-May-95

Syntax #DateFormat(*expression*)#

Argument	Required	Description
expression	Yes	String expression to be formatted.

Remarks

See Also **TimeFormat, Format**

Example

```
<webPrint>
DateFormat(CurrentDate()) = #DateFormat(CurrentDate())#
</webPrint>
```

TimeFormat Function

Description Displays a database date/time or time field using the HH:MM AM/PM format. 10:22 AM

Syntax #TimeFormat(*expression*)#

Argument	Required	Description
expression	Yes	String expression to be formatted.

Remarks

See Also **DateFormat, Format**

Example

```
<webPrint>
TimeFormat(CurrentTime()) = #TimeFormat(CurrentTime())#
</webPrint>
```

NumberFormat Function

Description Displays a numeric expression or a database numeric value as an integer with the thousand-separators. Eg. 1,234,567

Syntax #NumberFormat(*expression*)#

Argument	Required	Description
expression	Yes	Numeric expression to be formatted.

Remarks

See Also **Format**

Example

```
<webPrint>
```

*Please also refer to the examples in **Format** Function

```
NumberFormat(N1) = #NumberFormat(N1)#
</webPrint>
```

DecimalFormat Function

Description Displays a numeric expression or a database numeric value as an integer with the thousand-separators and two decimal places displayed. Eg. 1,234,567.90

Syntax #DecimalFormat(*expression*)#

Argument	Required	Description
expression	Yes	Numeric expression to be formatted.

Remarks

See Also **Format**

Example *Please also refer to the examples in **Format** Function

```
DecimalFormat(N1) = <webPrint> #DecimalFormat(N1)# </webPrint>
```

DollarFormat Function

Description Displays a numeric expression or a database numeric value as an integer with the dollar sign ($) in front, thousand-separators, and two decimal places. Negative values are enclosed with parentheses. Eg. $1,234,567.90

Syntax #DollarFormat(*expression*)#

Argument	Required	Description
expression	Yes	Numeric expression to be formatted.

Remarks

See Also **Format**

Example *Please also refer to the examples in **Format** Function

```
DollarFormat(N1) = <webPrint> #DollarFormat(N1)# </webPrint>
```

YesNoFormat Function

Description Displays boolean data as Yes or No. All non-zero values are displayed as Yes; zero values are displayed as No.

Syntax #YesNoFormat(*expression*)#

Argument	Required	Description

expression	Yes	Numeric expression to be formatted.

Remarks

See Also **Format**

Example *Please also refer to the examples in **Format** Function

```
YesNoFormat(N1) = <webPrint> #YesNoFormat(N1)# </webPrint>
```

ParagraphFormat Function

Description Converts CR/LF sequences into spaces and double CR/LF sequences into HTML paragraph markers (<P>). Useful for displaying data entered into TEXTAREA fields.

Syntax #ParagraphFormat(*expression*)#

Argument	Required	Description
expression	Yes	String expression to be formatted.

Remarks

See Also **StripCR Function**

Example

```
<webPrint>
Your text message is:  #ParagraphFormat(Form.MessageTextArea)#
</webPrint>
```

StripCR Function

Description Strips all carriage returns from the input string expression. Useful for pre-formatted (PRE) display of data entered into TEXTAREA fields.

Syntax #StripCR(*expression*)#

Argument	Required	Description
expression	Yes	String expression to be formatted.

Remarks This function is equivalent to #**StrReplace**(*stringexpr*, Chr(13),"")#

See Also **ParagraphFormat Function**

Example

```
<webPrint>
Your text message is:  #StripCR(Form.MessageTextArea)#
</webPrint>
```

HTMLCodeFormat Function

Description Strips carriage returns and escapes all special characters: greater than (>), less than (<), quote ('), and ampersand (&). Useful for display of HTML code posted using TEXTAREA fields.

Syntax #HTMLCodeFormat(*expression*)#

Argument	Required	Description
expression	Yes	String expression to be formatted.

Remarks

See Also **HTMLEditFormat**

Example

```
<webPrint>
Your text message is:  #HTMLCodeFormat(Form.MessageTextArea)#
</webPrint>
```

HTMLEditFormat Function

Description Behaves identically to **HTMLCodeFormat** except that it does not add the <PRE> tag to the text that is outputted.

Syntax #HTMLEditFormat(*expression*)#

Argument	Required	Description
expression	Yes	String expression to be formatted.

Remarks

See Also **HTMLCodeFormat**

Example

```
<webPrint>
Your text message is:  #HTMLEditFormat(Form.MessageTextArea)#
</webPrint>
```

ParameterExists Function

Description Accepts any syntactically valid parameter name and returns the value 1 (True) if the parameter has been passed to the current script or the value 0 (False) if the parameter has not been passed.

Syntax #ParameterExists(*formvariable*)#

Argument	Required	Description
formvariable	Yes	CGI form variable.

Remarks

See Also **HTMLCodeFormat**

Example For example, to test whether a form variable named 'UserID' has been passed to a script you would use the syntax:

```
<webIf #ParameterExists(Form.UserID)# >
```

PreserveSingleQuotes Function

Description Preserves single quotes (') in SQL statements. Used for writing SQL statements where you want to prevent Web+™ from automatically 'escaping' single-quotes contained in values derived from dynamic parameters.

Syntax #PreserveSingleQuotes(*expression*)#

Argument	Required	Description
expression	Yes	String expression.

Remarks

See Also

Example For example, to include a dynamic parameter in a SQL statement and suppress the escaping of single quotes you could use syntax like this:

```
SELECT * FROM Customers
WHERE CustomerName  IN ( #PreserveSingleQuotes(CustNames)# )
```

URLEncodedFormat Function

Description URL encodes (replace spaces with a '+' and non-alphanumeric characters with equivalent hexadecimal escape sequences) the string parameter which is passed to it. This function enables you to pass arbitrary strings (including those with spaces in them) within URLs (Web+™ will automatically decode all URL parameters which are passed to a script).

Syntax #URLEncodedFormat(*expression*)#

Argument	Required	Description
expression	Yes	String expression to be formatted.

Remarks

See Also

Example

```
<webPrint>
#URLEncodedFormat(URLvariable)#
</webPrint>
```

ValueList Function

Description Returns a comma (,) separated list of the values of each of the records from a given database query column. For example, if you ran a query which returned five distinct user-ids, the result of **ValueList function** would be something like: 12,33,45,51,66.

Syntax #ValueList(*querycolumn*)#

Argument	Required	Description
querycolumn	Yes	Name of the input database query column.

Remarks

See Also **QuotedValueList**

Example

```
<webPrint>
#ValueList(UserID)#
</webPrint>
```

QuotedValueList Function

Description Returns a quote enclosed ('xxx') and comma (,) separated list of the values of each of the records from a given database query column. For example, if you ran a query which returned five distinct user-ids, the result of **QuotedValueList** function would be something like: '12','33','45','51','66'.

Syntax #QuotedValueList(*querycolumn*)#

Argument	Required	Description
querycolumn	Yes	Name of the input database query column.

Remarks

See Also **ValueList**

Example

```
<webPrint>
#QuotedValueList(UserID)#
</webPrint>
```

ValidateCC Function

Description Returns a 0 value for an invalid credit card number, or a non-zero value represents different card type.

Syntax `#validatecc(numericexpression)#`

Argument	Required	Description
numericexpression	Yes	The argument can be any numeric expression.

The following table shows the output and the representation of the card type for the validatecc function:

Output	Card Type
0	Invalid credit card number
1	American Express
2	Visa
3	Master
4	Discover
9	Unknown card type

Remarks

See Also

Example

```
<webPrint>
#ValidateCC(creditcardnumber)#
</webPrint>
```

Operator Precedence

When several operations occur in an expression, each part is evaluated and resolved in a predetermined order. That order is known as operator precedence. Parentheses can be used to override the order of precedence and force some parts of an expression to be evaluated before others. Operations within parentheses are always performed before those outside. Within parentheses, however, normal operator precedence is maintained.

When expressions contain operators from more than one category, arithmetic operators are evaluated first, comparison operators are evaluated next, and logical operators are evaluated last. Within individual categories, operators are evaluated in the order of precedence shown below:

Arithmetic Operators (Listed by precedence)

Precedence	Operator	Description
1	`-`	Negation
2	`^`	Exponentiation
3	`*` `/`	Multiplication Division
3	`%`	Modulo arithmetic
4	`+` `-`	Addition Subtraction
5	`&`	String concatenation

*Note When multiplication and division occur together in an expression, each operation is evaluated as it occurs from left to right. Likewise, when addition and subtraction occur together in an expression, each operation is evaluated in order of appearance from left to right.

*Note The string concatenation operator (&) is not really an arithmetic operator, but in precedence it does fall after all arithmetic operators and before all comparison operators.

Comparison Operators (Equal precedence)

Precedence	Operator	Description
5	`=` `Eq`	Equality
5	`!=` `Ne`	Inequality
5	`Lt`	Less than. Conventional symbol: <
5	`Gt`	Greater than. Conventional symbol: >
5	`Lte`	Less than or Equal to. Conventional symbol: <=

5	**Gte**	Greater than or Equal to. Conventional symbol: >=
5	**?** **Contain** **Contains**	Contains
5	**!?** **NotContain**	Does not contain

*Note All comparison operators have equal precedence, that is, they are evaluated in the left-to-right order in which they appear.

*Note The Contain (?) and NotContain (!?) operators are not really comparison operators. They are actually a pattern-matching operators. They are equal in precedence to all comparison operators.

Logical Operators (Listed by precedence)

Precedence	Operator	Description
6	**Not, !**	Not
7	**And, &&**	And
8	**Or, ||**	Or

Operators

Description Operators are available for numbers n1 and n2 or strings s1 and s2. If any value in an expression is Null then the expression's value is Null. The order of operator evaluation is controlled by operator precedence.

Syntax * / % + - & Lt Lte Gt Gte = Eq != Ne Not And Or ? Contain !? NotContain

Operator	Description
n1 ***** n2	Multiply n1 by n2.
n1 **/** n2	Divide n1 by n2.
n1 **+** n2	Add n1 to n2.
n1 **-** n2	Difference of n1 and n2.
s1 **&** s2	Concatenate s1 with s2.
n1 **Lt** n2	Return True if n1 is less than n2. Conventional symbol: <
n1 **Lte** n2	Return True if n1 is less than or equal to n2. Conventional symbol: <=
n1 **Gt** n2	Return True if n1 is greater than n2. Conventional symbol: >
n1 **Gte** n2	Return True if n1 is greater than or equal to n2. Conventional symbol: >=
n1 **=** n2	Return True if n1 is equal to n2. Conventional symbol: =
n1 **Eq** n2	Return True if n1 is equal to n2. Conventional symbol: =
n1 **!=** n2	Return True if n1 is not equal to n2. Conventional symbol: <>
n1 **Ne** n2	Return True if n1 is not equal to n2. Conventional symbol: <>

`s1 = s2`	Return True if s1 is equal to s2.
`s1 Eq s2`	Return True if s1 is equal to s2.
`s1 != s2`	Return True if s1 is not equal to s2.
`s1 Ne s2`	Return True if s1 is not equal to s2.

Precedence When several operators are used in an expression, each operator is evaluated in a predetermined order. Operators are evaluated in this order:

- - (negate)
- ^ (exponentiation)
- * (multiply), / (divide)
- + (add), - (difference)
- & (string concatenate)
- = (equal), <> (not equal), < (less than) > (greater than), <= (less than or equal to), >= (greater than or equal to), Is (object equivalence)
- Not (logical not)
- And (logical and)
- Or (logical or)

Operators shown on the same line are evaluated from left to right.

Example

```
<webSet #N1# = 10 >
<webSet #N2# = 3 >
<webSet #S1# = "Talent" >
<webSet #S2# = "Soft" >

The initial value of the four variables that
demonstrate
the effects of operators are:<P>

<PRE>
<webPrint>

N1 = #N1#
N2 = #N2#
S1 = #S1#
S2 = #S2#

!   N1    = #!   N1#
Not N1    = #Not N1#
N1 * N2   = #N1 * N2#
N1 / N2   = #N1 / N2#
N1 % N2   = #N1 % N2#
N1 + N2   = #N1 + N2#
N1 - N2   = #N1 - N2#
S1 & S2   = #S1 & S2#
N1 Lt  (<)  N2  = #N1 Lt N2#
N1 Lte (<=) N2  = #N1 Lte N2#
N1 Gt  (>)  N2  = #N1 Gt N2#
N1 Gte (>=) N2  = #N1 Gte N2#
N1 =   N2  = #N1 =  N2#
N1 Eq  N2  = #N1 Eq N2#
N1 !=  N2  = #N1 != N2#
```

```
N1 Ne  N2  = #N1 Ne N2#
S1 =$  S2  = #S1 =$  S2#
S1 EqS S2  = #S1 EqS S2#
S1 !=$ S2  = #S1 !=$ S2#
S1 NeS S2  = #S1 NeS S2#
S1 Lt  S2  = #S1 Lt  S2#
S1 Lte S2  = #S1 Lte S2#
S1 Gt  S2  = #S1 Gt  S2#
S1 Gte S2  = #S1 Gte S2#
N1 &&  N2 = #N1 &&  N2#
N1 And N2 = #N1 And N2#
N1 ||  N2 = #N1 ||  N2#
N1 Or  N2 = #N1 Or  N2#

</webPrint>
</PRE>
```

^ Operator

Description Used to raise a number to the power of an exponent.

Syntax `# number ^ exponent #`
The number and exponent operands can be any numeric expression. When more than one exponentiation is performed in a single expression, the ^ operator is evaluated as it is encountered from left to right.

Remarks The result of ^ computation is a real number.

See Also Operator precedence.

Example
```
<webSet #Power# = #Number ^ Exponent# >
```

* Operator

Description Used to multiply two numbers.

Syntax `# operand1 * operand2 #`
The operands can be any numeric expression, ie. any sequence of characters that can be interpreted as a number, or any combination of variables, constants, functions, and operators that Web+™ can evaluate to a number.

See Also Operator precedence.

Example
```
<webSet #TotalAmount# = #(UnitPrice * Quantity)# >
<webSet #TotalProfit# = #((Cost + Markup) * Quantity)# >
```

/ Operator

Description Used to divide two numbers..

Syntax `# operand1 / operand2 #`
The operands can be any numeric expression, ie. any sequence of characters that can be interpreted as a number, or any combination of variables, constants, functions, and operators that Web+™ can evaluate to a number.

See Also Operator precedence.

Example
```
<webSet #AverageDailyRevenue# = #(AnualRevenue / 365)# >
```

% Operator

Description Divides two numbers and returns only the remainder.

Syntax `# operand1 % operand2 #`

Remarks The modulus, or remainder, operator divides *operand1* by *operand2* (rounding floating-point numbers to integers) and returns only the remainder as result. For example, in the expression `<webSet #A# = #(19.1 % 6.7)# >` , A (which is result) equals 5. The operands can be any numeric expression.

See Also Operator precedence.

Example
```
<webSet #A# = #(19.1 % 6.7)# >          returns A=5
```

+ Operator

Description Used to sum two numbers.

Syntax `# operand1 + operand2 #`
The operands can be any numeric expression, ie. any sequence of characters that can be interpreted as a number, or any combination of variables, constants, functions, and operators that Web+™ can evaluate to a number.

See Also Operator precedence.

Example
```
<webSet #TotalSales# = #(Region1Sales + Region2Sales +
Region3Sales)# >
```

- Operator

Description Used to subtract two numbers.

Syntax `# operand1 - operand2 #`
The operands can be any numeric expression, ie. any sequence of characters that can be interpreted as a number, or any combination of variables, constants, functions, and operators that Web+™ can evaluate to a number.

See Also Operator precedence.

Example
```
<webSet #Profit# = #(SalesPrice - Cost)# >
```

Comparison Operators

Comparison Operators:

= Eq

!= Ne

=$ EqS

!=$ NeS

Lt Gt Lte Gte

? Contain Contains

!? NotContain

Description Comparison operators, also known as relational operators, are used to compare two expressions.

Syntax # *expr1* operator *expr2* #

The operands can be any numeric expression, ie. any sequence of characters that can be interpreted as a number, or any combination of variables, constants, functions, and operators that Web+™ can evaluate to a number.

Operator	Description	True if	False if
{ **=** | **Eq** }	Equal to (=) Numeric or string (case-sensitive) comparison.	expr1 = expr2 expr1 Eq expr2	expr1 != expr2 expr1 Ne expr2
{ **!=** | **Ne** }	Not equal to (<>) Numeric or string (case-sensitive) comparison.	expr1 != expr2 expr1 Ne expr2	expr1 = expr2 expr1 Eq expr2
{ **=$** | **EqS** }	Equal to (=) Numeric or string (non case-sensitive) comparison.	expr1 =$ expr2 expr1 EqS expr2	expr1 !=$ expr2 expr1 NeS expr2
{ **!=$** | **NeS** }	Not equal to (<>) Numeric or string (non case-sensitive) comparison.	expr1 !=$ expr2 expr1 NeS expr2	expr1 =$ expr2 expr1 EqS expr2

Lt	Less than (<)	expr1 Lt expr2	expr1 Gte expr2
Gt	Greater than (>)	expr1 Gt expr2	expr1 Lte expr2
Lte	Less than or Equal to (<=)	expr1 Lte expr2	expr1 Gt expr2
Gte	Greater than or Equal to (>=)	expr1 Gte expr2	expr1 Lt expr2
{ **?** | **Contain** | **Contains** }	Contains		
{ **!?** | **NotContain** }	Does not contain		

See Also Operator precedence.

Example

```
<webIf #Age Lt 21# >
  <webSet #Message# = "You are not old enough to view this
page.">
</webIf>
```

Logical Operators

&& And Operator

Description Used to perform a logical conjunction on two expressions.

Syntax # expr1 { && | And } expr2 #

The operands can be any numeric expression, ie. any sequence of characters that can be interpreted as a number, or any combination of variables, constants, functions, and operators that Web+™ can evaluate to a number.
If, and only if, both expressions evaluate True, result is True. If either expression evaluates False, result is False. The following table illustrates how result is determined:

If expr1 is	And expr2 is	The result is
True	True	True
True	False	False
True	Null	Null
False	True	False
False	False	False
False	Null	False
Null	True	Null
Null	False	False
Null	Null	Null

See Also Operator precedence.

Example
```
<webIf #Age Gt 21 && Gender = "Male"# >
```

|| Or Operator

Description Used to perform a logical disjunction on two expressions.

Syntax # expr1 { || | Or } expr2 #

The operands can be any numeric expression, ie. any sequence

of characters that can be interpreted as a number, or any combination of variables, constants, functions, and operators that Web+™ can evaluate to a number.
If, and only if, both expressions evaluate True, result is True. If either expression evaluates False, result is False. The following table illustrates how result is determined:

If expr1 is	And expr2 is	The result is
True	True	True
True	False	True
True	Null	True
False	True	True
False	False	False
False	Null	Null
Null	True	True
Null	False	Null
Null	Null	Null

See Also Operator precedence.

Example
```
<webIf #(sales Gt 5000) Or (hoursWorked Gt 81)# >
```

! Not Operator

Description Used to perform logical negation on an expression.

Syntax
```
# { ! | Not } expr #
```
The operands can be any numeric expression, ie. any sequence of characters that can be interpreted as a number, or any combination of variables, constants, functions, and operators that Web+™ can evaluate to a number.

If expr is	The result is
True	False
False	True
Null	Null

See Also Operator precedence.

Example
```
#"PC" != "pc"#
```

& String Concatenation Operator

Description Used to concatenate two strings.

Syntax `# operand1 & operand2 #`
The operands can be any numeric expression, ie. any sequence of characters that can be interpreted as a number, or any combination of variables, constants, functions, and operators that Web+™ can evaluate to a number.
Whenever an operand is a number, it is converted to a string.

See Also Operator precedence.

Example
```
<webSet #Name# = #(LastName & FirstName)# >
```

? Like Operator (replaces Contain)

Description Used to compare two string expressions.

Syntax `# expression { ? | Like } pattern #`
If expression contains pattern, result is True; if there is no match, result is False. The operands can be any numeric expression, ie. any sequence of characters that can be interpreted as a number, or any combination of variables, constants, functions, and operators that Web+™ can evaluate to a number.

If expression matches pattern, result is 1 (True); if there is no match, result is 0 (False). The case sensitivity and character sort order of the Like operator depend on the setting of the Option Compare statement. Unless otherwise specified, the default string-comparison method for each module is Option Compare Binary; that is, string comparisons are case-sensitive.

Built-in pattern matching provides a versatile tool for string comparisons. The pattern-matching features allow you to use wildcard characters, such as those recognized by the operating system, to match strings. The wildcard characters and what they match are shown in the following table:

See Also **!? NotLike**

Example
```
<webIf #Name ? "jo"# >
```

!? NotLike Operator (replaces NotContain)

Description Used to compare two string expressions.

Syntax `# expression { !? | NotLike } pattern #`
If expression contains pattern, result is True; if there is no match, result is False. The operands can be any numeric expression, ie. any sequence of

characters that can be interpreted as a number, or any combination of variables, constants, functions, and operators that Web+™ can evaluate to a number.

See Also **? Like**

Example

```
<webIf #Name !? "jo"# >
```

Flow Control Statements

webAbort Statement

Description Causes the currently executing script to terminate immediately.

Syntax <**webAbort**>

Remarks

See Also

Example <**webAbort**>

webIf…webElseIf…webElse Statement

Description Allows conditional execution, based on the evaluation of an expression.

```
Syntax      <webIf #condition1#>
              [statementblock-1]
            [<webElseIf #condition2#>
              [statementblock-2]]
            [<webElse>
              [statementblock-n]]
            </webIf>
```

Part	Required	Description
<webIf>	Yes	Keyword that begins the block <**webIf**>...</**webIf**> decision control structure.
condition1	Yes	One of two types of expressions: A numeric or string expression that evaluates true (nonzero) or false (0 and Null).
statementblock-1	No	One or more Web+ statements executed if *condition1* is true.
<ElseIf>	No	Keyword indicating that alternative conditions must be evaluated if condition1 is not satisfied.
condition2	No	Same as *condition1* used in the single-line form shown above.
statementblock-2	No	One or more Web+ statements executed if *condition2* is true.
<webElse>	No	Keyword used to identify the actions taken if none of the previous conditions are satisfied.
statementblock-n	No	One or more Web+ statements executed if *condition1* and

		condition2 are both false.
</webIf>	Yes	Keyword that ends the block form of the <**webIf**>...</**webThen**>.

The <**webElse**> and <**webElseIf**> clauses are both optional. You can have as many <**webElseIf**> clauses as you like in a block <**webIf**>, but none can appear after an <**webElse**> clause. Any of the statement blocks can contain nested block <**webIf**> statements.

Remarks **webSelectCase…webCase** may be more useful when evaluating a single expression that has several possible actions.

See Also **webSelect…webCase…webCaseElse**

Example

```
<webIf #I Lt 10#>
  <webPrint> I is less than 10 </webPrint>
<webElseIf #I = 10#>
  <webPrint> I is equal to 10 </webPrint>
<webElse>
  <webPrint> I is larger than  10 </webPrint>
</webIf>
```

webSelectCase…webCase…webCaseElse Statement

Description Executes one of several statement blocks depending on the value of an expression.

Syntax

```
<webSelectCase testexpression>
  [ <webCase expressionlist1>
            [statementblock-1] ]
  [ <webCase expressionlist1>
            [statementblock-2] ]
  [ <webCaseElse>
            [statementblock-n] ]
</webSelectCase>
```

The Select Case syntax has these parts:

Part	Required	Description
<webSelectCase>	Yes	Begins the Select Case decision control structure. Must appear before any other part of the Select Case structure.
testexpression	Yes	Any numeric or string expression. If testexpression matches the expressionlist associated with a Case clause, the statementblock following that Case clause is executed up to the next Case clause, or for the final one, up to the End Select. Control then passes to the statement following End Select. If testexpression matches more than one Case clause, only the statements following the first match are executed.
<webCase>	Yes	Sets apart a group of Web+ statements to be executed if an expression in expressionlist matches testexpression.
expressionlist	No	One or more Web+ statements executed if *condition1* is true.
statementblock-1	No	Elements *statementblock-1* to *statementblock-n* consist of any number of Web+ statements on one or more lines.
<webCaseElse>	No	Keyword indicating the statementblock to be executed if no

		match is found between the testexpression and an expressionlist in any of the other Case selections. When there is no Case Else statement and no expression listed in the Case clauses matches testexpression, program execution continues at the statement following End Select.
statementblock-n	No	One or more Web+ statements executed if *condition1* and *condition2* are both false.
</webSelectCase>	Yes	Ends the **<webSelectCase>**. Must appear after all other statements in the **<webSelectCase>** control structure.

The <**webCaseElse**> clause is optional. You can have as many <**webCase**> clauses as you like in a block <**webSelectCase**>, but none can appear after an <**webCaseElse**> clause. Any of the statement blocks can contain nested block <**webSelectCase**> statements.

Remarks Although not required, it is a good idea to have a <**webSelectCase**> statement in your Select Case block to handle unforeseen testexpression values.

See Also **webIf…webElseIf…webElse.../webIf**

Example Example 1: Nested Select...Case

```
<webSet #D#=1>
<webSelectCase #I#>
<webCase #D#>
  This is Case 1
  <webSet #J# = "ABXC">
  <webSelectCase #J#>
  <webCase "AB"> <BR> This is the nested select1
  <webCaseElse> This is Other!!!
  </webSelectCase>
<webCaseElse>
  This is Case 2
  <webSet #J# = "AB">
  <webSelectCase #J#>
    <webCase "AB"> <BR> This is the nested select2
  </webSelectCase >
</webSelectCase >
```

Example 2: Conditional expressions

```
<webSelectCase #X+Y*(Z+1)#>
<webCase #I+4#>
...
<webCaseElse>
...
</webSelectCase >
```

webFor…webExitFor Statement

Description Repeats a group of instructions a specified number of times.

Syntax

```
<webFor #counter# = #startvalue# To #endvalue# [Step
#incrementvalue#]>
    [statementblock]
       [<webExitFor>]
    [statementblock]
```

</webFor [*counter*]>

The webFor statement has these parts:

Part	Required	Description
<webFor>	Yes	Begins a For loop control structure. Must appear before any other part of the structure.
counter	Yes	Numeric variable used as the loop counter.
startvalue	Yes	Initial value of *counter*.
To	Yes	Separates *startvalue* and *endvalue*.
endvalue	Yes	Final value of *counter*.
Step	No	Indicates that increment is explicitly stated. The **Step** value controls loop execution as follows: When **Step** value is positive or 0, loop executes if *counter* is less than or equal to (<=) *endvalue*. When **Step** value is negative, loop executes if *counter* is greater than or equal to (>=) *endvalue*.
incrementvalue	No	Amount *counter* is changed each time through the loop. If you do not specify **Step**, *incrementvalue* is default to 1.
statementblock	No	Program lines between <**webFor**> and </**webFor**> that are executed the specified number of times.
<webExitFor>	No	Only used within a **webFor** control structure to provide an alternate way to exit. Any number of **webExitFor** statements may be placed anywhere in the **webFor** loop. Often used with the evaluation of some condition (for example, **webIf**), **webExitFor** transfers control to the statement immediately following the </**webFor**> tag.
</webFor>	Yes	Ends a **webFor** loop. Causes *incrementvalue* to be added to *counter*. The </**webFor**> tag may include an optional *counter* name for better code readability, such as: </**webFor** I>

Once the loop has been entered and all the statements in the loop have executed, **Step** is added to *counter*. At this point, either the statements in the loop execute again (based on the same test that caused the loop to execute in the first place), or the loop is exited and execution continues with the statement following the </**webFor>** statement.

You can nest **webFor** loops by placing one **webFor** loop within another. Give each loop a unique variable name as its counter. The following construction is correct:

```
<webFor #I# = 1 To 10 Step 1>
  <webFor #J# = 1 To 10>
    <webFor #K# = 1 To 10>
      ...
    </webFor K>
  </webFor J>
</webFor I>
```

Remarks Changing the value of *counter* while inside a loop can make the program more difficult to read and debug.

See Also **webWhile**…**webExitWhile**

Example

```
<webSet #startvalue# = 10 >
<webSet #endvalue# = 20 >
```

```
<webSet #incrementvalue# = 2>

<webFor #I# = #startvalue# To #endvalue# Step #incrementvalue#>
  <webPrint> I = #I# </webPrint>

  <webFor #J# = 1 To 10>
    <webPrint> J = #J# </webPrint>

    <webFor #K# = 1 To 10>
      <webPrint> K = #K# </webPrint>
      <webIf #K#=5>
        <webExitFor K>
      </webIf>
      ...
    </webFor K>
  </webFor J>
</webFor I>
```

webWhile…webExitWhile Statement

Description Repeats statements between the `<webWhile value operator value>` and `</webWhile>` statements while a given condition is True.

Syntax

```
<webWhile #value operator value#>
    statements
    [<webExitWhile>]
    statements
</webWhile>
```

See Also **webFor…webExitFor**

Example

```
<webSet #I#=1>
<webWhile #I Lt 4# >
  <webPrint>
    This is loop #I#<BR>
  </webPrint>
  <webSet #I# = #(I + 1)#>
</webWhile>
```

The *value* elements in the webIf, webElseIf, webWhile, webSelectCase, and webCase tag can be of three different types:

Value Type	Explanation	Examples
Dynamic parameter	Form, URL, and CGI parameters as well as fields from query result sets.	#Form.Name#, #URL.Age# #CGI.User_Agent#
Literal numeric values	Integer values (not delimited by quotes).	0, 22, 130
Literal string values	Any sequence of characters (delimited by quotes).	"Mozilla", "User Guide", "Authorized"

The *operator* element in the webIf, webElseIf, webWhile, webSelectCase, and webCase tag must be one of the following:

Operator	Explanation
= **Eq**	Performs a case-insensitive comparison of the two values and returns true if they are identical.

<> **Ne**	Opposite behavior of *is*.
? **Contain**	Checks to see if the value on the left is contained in the value on the right and returns true if it is.
!? **Does not contain**	Opposite behavior of *contains*.
Gt	Checks to see if the value on the left is greater than the value on the right and returns true if it is.
Lt	Opposite behavior of *greater than*.
Gte	Checks to see if the value on the left is greater than or equal to the value on the right and returns true if it is.
Lte	Checks to see if the value on the left is less than or equal to the value on the right and returns true if it is.

webContent Statement

Description Defines the content-type being returned by the current script.

Syntax <**webContent Type**="*ContentType*">
Where *ContentType* defines the content-type being returned by the current script.

Remarks None.

See Also **webLocation, webInclude, webContent**

Example <**webContent Type**="*ContentType*">

webLocation Statement

Description Redirects the browser request to another URL. Allow the current browser request to refer another document Instead of outputting the document, you can just tell the browser where to get the new one, or have the server automatically output the new one for you. Defines the path to an HTML file or WML script.

Syntax <**webLocation URL**="*URLAddress*">
Where *URLAddress* defines the target URL address.

Remarks None.

See Also **webInclude**

Example
```
<webLocation
URL="http://www.talentsoft.com/webplus/registration.htm/">
```

webInclude Statement

Description Allows the current script to call and include an external script file for current use. The external script file will be processed as if it is a part of the current script file into which it is included.

Syntax <**webInclude Script**=" *script*`FileName`">
Where *ScriptFileName* defines the logical path to an existing WML script file.

Remarks None.

See Also **webLocation**

Example <**webInclude Script**="/script_library/footer.wml">

Program Execution Statements

webRun Statement

Description Runs an executable program.

Syntax

```
<webRun Exe=#ExecutableCommandLineString#
[Type="ExecutableType"] [InPara="InputParameter"]
[OutVar=#OutputVariableName#] [TimeOut=NumberSeconds]
[Password=WebRunPassword]>
```

Argument	Required	Description
Exe	Yes	Specifies the name of the executable program and its command line arguments. Remarks In the Windows environment, the executable file should have an extention of .exe, .com, .bat, or .cmd
Type	No	Executable program type. Possible values: "**Win32**" = 32-bit Windows executable "**Win16**" = 16-bit Windows executable * For UNIX executables, this argument may be blank ("") or omitted.
InPara	No	Specifies the input parameters to be passed to passed the executable program.
OutVar	No	Specifies the name of the output variable which will store the result (standard output) of the executable program. Remarks Since the executable program cannot pass standard output directly to Web+™, this output variable is used to store the output of the executable program which can then be read or used by Web+™.
TimeOut	No	Specifies the number of seconds which webRun will wait before finishing and closing the current script file. Remarks It means that the creator (parent process) of the webRun's exe file should wait for that period of time for the child process to finish. If time exceeds, the parent process will receive a notice about timeout and continue to execute. This is useful when the parent want to use the result of the child process. You can have the timeout to be INFINITE so the Parent will wait for the child forever(I will add this function soon)
Password	No	Specifies the webRun Password in the webpsvr.ini file under the

		[WebRun] Section. Remarks This webRun password is optional; if you did not type in anything for the webRunPassword under the [webRun] Section, webRun statement does not requires any password to proceed.

Remarks

See Also

Example

```
<webRun Exe="test.exe 1 test.dat" Type="win32" OutFile="out.001"
TimeOut=100>
```

SMTP Mail Statements

webMail Statement

Description Sends SMTP mail.

Syntax

```
<webMail [From="SenderEmailAddress"] To="RecepientEmailAddress"
[Cc="CopyToEmailAddresses"][Attach="FileName"] [SendMode="Text|
Binary"] [SUBJECT=Subject] [Type="Type"] [Query="QueryName"]
[MaxRows="MaximumNumberOfRows"] [Group="SetOfRecords"]
[Server="SMTPServerType"] [Port="TCP/IPProtOnSMTPServer"]
[TimeOut=NumberOfSeconds]>
```

Argument	Required	Description
To	Yes	The name of the recipient(s) of the e-mail message. This can be either a static address (e.g., TO="support@TalentSoft.com"), a variable containing an address (e.g., TO="#Form.Email#") or the name of a query column which containing address information (e.g., TO="#EMail#"). In the latter case, an individual e-mail message will be sent for every row returned by the query. The TO attribute is required.
From	No	The sender of the e-mail message. This attribute may either be static (e.g. FROM="support@TalentSoft.com") or dynamic (e.g. FROM="#GetUser.EMailAddress#"). The FROM attribute is required.
Cc	No	Additional addresses to copy the e-mail message to (this attribute may also be static or dynamic). The CC attribute is optional.
Attach	No	Attachment for file is also allowed. You can write the Attach argument with variable or string representing the file's logical path.
SendMode	No	Optional attribute defines the attached file mode: either a Text file or a Binary file. If SendMode does not specified, the default mode is Text.
Type	No	Optional attribute specifying extended type attributes for the message. Currently, the only valid value for this attribute is "HTML". Specifying TYPE="HTML" informs the receiving e-mail client that the message has embedded HTML tags that need to be processed. This is only useful when sending messages to mail clients understand HTML (e.g., the Netscape 2.0 e-mail client). Remarks Since the executable program cannot pass output directly to Web+™, this output file is used to store the output of the executable program which can then be read or used by Web+™.
Query	No	The name of the webQUERY from which you want to draw data for message(s) to be sent. This attribute is optional (specify it if you want to send more than one mail message or if you want to

		send the results of a query within a single message).
MaxRows	No	Optional attribute specifying the maximum number of mail messages you want to send.
Group	No	Optional attribute specifying the query column to use in grouping sets of records together for sending as a single mail message. For example, if you are sending a set of billing statements out to your customers, you might group on 'Customer_ID'.
Server	No	The address of the SMTP server to use for sending messages. Remarks This optional attribute is used to override the default SMTP Server settings created with the Web+™ Administrator.
Port	No	The TCP/IP port on which the SMTP server is listening for requests (this is almost always 25) Remarks This optional attribute is used to override the default SMTP Server settings created with the Web+™ Administrator.
TimeOut	No	The number of seconds to wait before timing out the connection to the SMTP server. Remarks This optional attribute is used to override the default SMTP Server settings created with the Web+™ Administrator.

Remarks

.

See Also

Example

```
<webMail QUERY="RecentForumsRequests"
        FROM="webmaster@TalentSoft.com"
        TO="marketing@TalentSoft.com"
        ATTACH="/webplus/Talentsoft.dat"
        SUBJECT="Web+ Sales Report">

Here is a list of people who have inquired about Web+™ Forums
over the last 7 days:

<webPrint>
#FirstName# #LastName# (#Company#) - #EMailAddress#
</webPrint>

Regards,

The WebMaster
webmaster@TalentSoft.com

</webMail>
```

File Processing Statements

webFileOpen Statement

Description Opens a file for input/output (read/write) processing.

Syntax

```
<webFileOpen Name=FileName Mode="{Read|Write|Append}"
[Lock="{Shared|Read|Write|Read Write}"] As=FileAliasName
Ret=#FileReturnCodeVariableName# [Password=ReadPassword|
WritePassword]>
```

Argument	Required	Description
Name	Yes	A string expression representing the logical path and name of the file to open. A path relative to the current directory can be used.
Mode	Yes	This argument specifies one of the following file open modes. One of the following three values must be specified. Possible Values: "**Read**" = Sequential read (input) input mode. Opens the file for reading only. "**Write**" = Sequential write (output) mode. Opens the file for writing only. "**Append**" = Sequential output mode. Append sets the file pointer to the end of the file. A `webFilePutChar` or `webFilePutLine` statement then extends (appends to) the file.
Lock	No	This argument specifies the file locking option in a multiuser or multiprocessing environment to restrict access by other processes or users to an open file. When Open is restricted by a previous process, a Permission denied error occurs. Possible Values: "" = If you don't specify a lock type, the file can be opened for reading and writing any number of times by the statement, but other operations are denied access to the file while it is open. "**Shared**" = Any process on any machine can read from or write to this file. "**Read**" = No other process is granted read access to this file. This lock is applied only if no other process has a previous read access to the file. "**Write**" = No other process is granted write access to this file. This lock is applied only if no other process has a previous write access to the file. "**Read Write**" = No other process is granted either read or write access to this file. This access is granted only if read or write access has not already been granted to another process, or if a Lock Read or Lock Write is not already in place.
As	Yes	A string expression representing the alias name of the file being opened. This alias name will be referred to in other webFile operations.

Ret	Yes	This argument specifies the name of the variable which stores the result of the webFileOpen action. Possible Values of the Variable: "**1**" = True, file open successful. "**0**" = False, file open unsuccessful.
Password	No	This argument specifies the access password for the file open operation. Possible Values of the Variable: If **Mode**= "**Read**", Password provided must be the FileReadPassword as set in webpsvr.ini. If **Mode**= "**Write**" or "**Append**", Password provided must be the FileWritePassword as set in webpsvr.ini. Remarks This password is optional; if you do not specify any password under the [File IO] section in the webpsvr.ini file, then this statement does not require any password to process.

Remarks You must open a file before any I/O operation can be performed on it. webFileOpen allocates a buffer for I/O to the file and determines the mode of access used with the buffer.

See Also <**webFileClose**>

Example

```
<webFileOpen NAME="/test.txt" MODE="WRITE" AS="FileExample"
RET=#FileReturnCode# [Password="WritePassword"]>
  <webSet #I# = 0>
  <webWhile #FileReturnCode != "EOF"#>
    <!-- Set the value for a variable: NewLine  -->
    <webSet #NewLine# = "Hello webPager!  This is my first
pager">
    <!-- Write a line to the text file -->
    <webFilePutLine FILE="FileExample" VALUE=#NewLine#
[Password="WritePassword"]>
    <webFilePutLine FILE="FileExample" VALUE=#NewLine#
[Password="WritePassword"]>
    <webSet #NewCharacter# = "A">
    <webFilePutChar FILE="FileExample" VALUE=#NewCharacter#
[Password="WritePassword"]>
    <webSet #I# = #I + 1#>
      <webPrint>
        Line #I# : #Message1#  <BR>
      </webPrint>
  </webWhile>
<!-- Close the text file -->
<webFileClose FILE="FileExample">
```

webFileClose Statement

Description Closes a file opened with the <**webFileOpen**>.

Syntax <**webFileClose File=***FileAliasName*>

Remarks If the *FileAliasName* parameter is omitted then all open files for the current script are closed.

See Also <**webFileOpen**>

Example

```
<webFileClose FILE="FileExample">
```

webFileCopy Statement

Description Copies a source file to a destination file.

Syntax

```
<webFileCopy From=SourceFileName To=DestinationFileName
Ret=#FileReturnCodeVariableName# [FromPassword=ReadPassword]
[ToPassword=WritePassword]>
```

Argument	Required	Description
From	Yes	A string expression representing the path and name of the source file to copy from.
To	Yes	A string expression representing the path and name of the destination file to copy to.
Ret	Yes	This argument specifies the name of the variable which stores the result of the webFileCopy action. Possible Values of the Variable: "True" = file copy successful. "False" = file copy unsuccessful.
FromPassword	No	This argument specifies the access password for the file open operation. Password provided must be the FileReadPassword as set in webpsvr.ini. Remarks This password is optional; if you do not specify any password under the [File IO] section in the webpsvr.ini file, then this statement does not require any password to process.
ToPassword	No	Password provided must be the FileWritePassword as set in webpsvr.ini. Remarks This password is optional; if you do not specify any password under the [File IO] section in the webpsvr.ini file, then this statement does not require any password to process.

See Also **webFileDelete, webFileRename**

Example

```
<webFileCopy From="SourceFileName" To="DestinationFileName"
Ret=#FileReturnCodeVariableName# [FromPassword=ReadPassword]
[ToPassword=WritePassword] >
```

webFileRename Statement

Description Renames a file.

Syntax

```
<webFileRename From=SourceFileName To=DestinationFileName
Ret=#FileReturnCodeVariableName# [FromPassword=ReadPassword]
[ToPassword=WritePassword]>
```

Argument	Required	Description
From	Yes	A string expression representing the path and name of the source

		file to be renamed.
To	Yes	A string expression representing the path and name of the destination file to rename to.
Ret	Yes	This argument specifies the name of the variable which stores the result of the webFileRename action. Possible Values of the Variable: "True" = file rename successful. "False" = file rename unsuccessful.
FromPassword	Yes	Password provided must be the FileReadPassword as set in webpsvr.ini. Remarks This password is optional; if you do not specify any password under the [File IO] section in the webpsvr.ini file, then this statement does not require password to process.
ToPassword	Yes	Password provided must be the FileWritePassword as set in webpsvr.ini.

See Also **webFileCopy, webFileDelete**

Example
```
<webFileRename From="SourceFileName" To="DestinationFileName"
Ret=#FileReturnCodeVariableName# FromPassword=ReadPassword
ToPassword=WritePassword >
```

webFileDelete Statement

Description Deletes a file.

Syntax
```
<webFileDelete Name=FileName [Ret=#ReturnCode#]
[Password=WritePassword]>
```

See Also **webFileCopy, webFileRename=**

Example
```
<webFileDelete Name="MyFile.txt" Ret=#ReturnCode#
Password="abc">
```

webFileGetChar Statement

Description Reads the current character from a file opened with the <**webFileOpen**>.

Syntax
```
<webFileGetChar File=AliasFileName Result=#ResultString#
[Password=ReadPassword]>
```

Argument	Required	Description
File	Yes	The alias name of the file being opened, define by the As argument in webFileOpen.
Result	Yes	When meet the end of file, a value "EOF" is stored in the variable defined by the Ret=... of webFileOpen, or a value of "NOEOF" is stored.

		Possible Values: "True" = file open successful. "False" = file open unsuccessful.
Password	Yes	Password provided must be the FileReadPassword as set in webpsvr.ini. Remarks This password is optional. If you did not specify any password in the [File IO] section, webFileGetChar does not require password to prcoess.

See Also **webFileGetLine**

Example
```
<webFileGetChar File=#SymbolicFileName# Result=#ResultString#
Password="abc">
```

webFileGetLine Statement

Description Reads the current line from a file opened with the <**webFileOpen**>.

Syntax
```
<webFileGetLine File=AliasFileName Result=#ResultString#
[Password=ReadPassword]>
```

See Also **webFileGetChar**

Example
```
<webFileGetLine File=#SymbolicFileName# Result=#ResultString#
Password="abc">
```

webFilePutChar Statement

Description Writes a character to a file opened with the <**webFileOpen**>.

Syntax
```
<webFilePutChar File=AliasFileName Value=#ChracterString#
[Password=WritePassword]>
```

See Also **webFilePutLine**

Example
```
<webFilePutChar File=#SymbolicFileName# Value=#ResultString#
Password="abc">
```

webFilePutLine Statement

Description Writes a line to a file opened with the <**webFileOpen**>.

Syntax
```
<webFilePutLine File=AliasFileName Value=#ChracterString#
[Password=WritePassword]>
```

See Also **webFilePutLine**

Example
```
<webFilePutChar File=#SymbolicFileName# Value=#ResultString#
Password="abc">
```

webGetProfile Statement

Description Reads the profile parameter value from a text file, usually a *.INI file with key(parameter) names and their values.

Syntax

```
<webGetProfile Section=SectionName Key=KeyName
Result=#ResultString# [Size=#BufferSize#] File=#FileName#
[Password=ReadPassword]>
```

Argument	Required	Description
Section	Yes	Profile section name, indicated by square brackets in the INI file. eg. [Main]
Key	Yes	Profile key name, or parameter name.
Result	Yes	Result, or value, of the profile key parameter.
Size	No	Buffer size. If Size is not specified, the default value is 1024 byte. If Result is longer than Size, anything that exceeds Size limit will be truncated.
File	Yes	File name of the INI file.
Password	Yes	Password provided must be the FileReadPassword as set in webpsvr.ini. Remarks This password is optional. If you did not specify any password in the [File IO] section, webGetprofile does not require password to prcoess.

See Also **webGetProfile**

Example

```
<webGetProfile Section=#SectionName# Key=#KeyName#
Result=#ResultString# Size=#BufferSize# File=#FileName#
Password="abc">
```

webPutProfile Statement

Description Writes the profile key (parameter) value to a text file, usually a *.INI file with key(parameter) names and their values.

Syntax

```
<webPutProfile Section=SectionName Key=KeyName
Value=#ValueString# File=#FileName# [Password=WritePassword]>
```

Argument	Required	Description
Section	Yes	Profile section name, indicated by square brackets in the INI file. eg. [Main]
Key	Yes	Profile key name, or parameter name.
Value	Yes	Value, of the profile key parameter to be written (put) to the file.
Size	No	Buffer size. If Size is not specified, the default value is 1024 byte. If Result is longer than Size, anything that exceeds Size limit will be truncated.
File	Yes	File name of the INI file.
Password	Yes	Password provided must be the FileWritePassword as set in

		webpsvr.ini. Remarks This password is optional. If you did not specify any password in the [File IO] section, webFilePutProfile does not require any password to prcoess.

See Also **webGetProfile**

Example
```
<webPutProfile Section=#SectionName# Key=#KeyName#
Result=#ResultString# Size=#BufferSize# File=#FileName#
Password="abc">
```

webPrint Statement

Description Prints out the file as a text file.

Syntax
```
<webPrint [Query=QueryName] [File=FileName]
[MaxRows=MaximunNumberOfRows] [Group=GroupByName]
[Password=ReadPassword] [HTMLFormat=On|Off] [PreFormat=On|Off]>
…
</webPrint>
```

Argument	Required	Description
File	Yes	The logical path of the file defined in the webpsvr.ini.
Password	No	Password provided must be the FileWritePassword as set in webpsvr.ini. Remarks This password is optional. If you did not specify any password in the [File IO] section, webFilePutProfile does not require any password to prcoess.
HTMLFormat	No	This option allows you to output the file in the browser. By default, it is On.
PreFormat	No	This option allows you to output the file "as it is" to the browser.

Example
```
<webPrint File="/webplus/webPrintExample.wml" Password="abc">
</webPrint>
```

Database Processing Statements

webDbInsert Statement

Description Inserts a new record into a database.

Syntax

```
<webDbInsert DataSource=ODBCDataSourceName
TableName=DatabaseTableName [TableOwner=TableOwnerName]
[TableQualifier=TableQualifier] [FormFields=Formfields] >
```

Argument	Required	Description
DataSource	Yes	Name of the ODBC data source containing your table.
TableName	Yes	Name of the table you want the form fields inserted to.
TableOwner	No	For data sources that support table ownership (e.g., SQL Server, Oracle, & Watcom SQL), you may use this field to specify the owner of the table.
TableQualifier	No	For data sources that support table qualifiers, you may use this field to specify the qualifier for the table. The purpose of table qualifiers varies across drivers (e.g., for SQL Server and Oracle the qualifier refers to the name of the database containing the table; for the Intersolv dBase driver the qualifier refers to the directory where the DBF files are located).
FormFields	No	A comma separated list of form fields to insert or update (if this attribute is not specified then all fields in the form are included in the operation).

Remarks The **TableOwner** and **TableQualifier** fields are rarely needed for most databases. They are provided for compatibility with ODBC drivers that require you to specify a table owner and/or table qualifier. Neither of these fields need to be specified for the Microsoft ODBC Desktop Drivers bundled with Web+™ (i.e. drivers for Access, Paradox, dBase, FoxPro, Excel, & Text). ODBC drivers that require table owners and/or qualifiers to be specified include all SQL Server and Oracle drivers as well as all Intersolv Q&E drivers.

See Also **webDbUpdate**

Example

```
<webDbInsert DataSource="Registration DB" TableName="User">
<HTML>
<Body>
<!-- Set the value for a variable: NewLine  -->
You have been registered for TalentSoft Internet Seminar.
</Body>
</HTML>
```

webDbUpdate Statement

Description Updates an existing record in a database.

Syntax
```
<webDbUpdate DataSource=ODBCDataSourceName
TableName=DatabaseTableName [TableOwner=TableOwnerName]
[TableQualifier=TableQualifier] [FormFields=Formfields] >
```

Argument	Required	Description
DataSource	Yes	Name of the ODBC data source containing your table.
TableName	Yes	Name of the table you want the form fields inserted to.
TableOwner	No	For data sources that support table ownership (e.g., SQL Server, Oracle, & Watcom SQL), you may use this field to specify the owner of the table.
TableQualifier	No	For data sources that support table qualifiers, you may use this field to specify the qualifier for the table. The purpose of table qualifiers varies across drivers (e.g., for SQL Server and Oracle the qualifier refers to the name of the database containing the table; for the Intersolv dBase driver the qualifier refers to the directory where the DBF files are located).
FormFields	No	A comma separated list of form fields to insert or update (if this attribute is not specified then all fields in the form are included in the operation).

Remarks The **TableOwner** and **TableQualifier** fields are rarely needed for most databases. They are provided for compatibility with ODBC drivers that require you to specify a table owner and/or table qualifier. Neither of these fields need to be specified for the Microsoft ODBC Desktop Drivers bundled with Web+™ (i.e. drivers for Access, Paradox, dBase, FoxPro, Excel, & Text). ODBC drivers that require table owners and/or qualifiers to be specified include all SQL Server and Oracle drivers as well as all Intersolv Q&E drivers.

See Also **webDbInsert**

Example
```
<webDbUpdate DataSource="Registration DB" TableName="User"
FormFields="PhoneNo">
<HTML>
<Body>
<!-- Set the value for a variable: NewLine  -->
Your registration for TalentSoft Internet Seminar has updated.
</Body>
</HTML>
```

webDbQuery Statement

Description Submits a query using an SQL statement to a database.

Syntax
```
<webDbQuery Name="Query Name" DataSource="ODBC DataSource Name"
SQL="SQL Statement" [MaxRows=MaximunNumberOfRows]
[TimeOut=Milliseconds] [Debug] >
```

Argument	Required	Description
Name	Yes	A name you assign to the query. Query names must begin with a letter and may consist of letters, numbers, and the underscore character (spaces are allowed if the query name is inside the double- quotes ("")). The query name is used later in the script to reference the queries result set.
DataSource	Yes	The name of the ODBC Data Source from which this query should retrieve data.
SQL	Yes	An SQL statement to be processed by the database.
MaxRows	No	The maximum number of records (rows) to be returned in the result set.
TimeOut	No	The maximum time in milliseconds for the query to execute before returning an error indicating that the query has timed-out. Note that this attribute is not supported by most ODBC drivers (it is, however, supported by the SQL Server 6.0 driver).
Debug	No	Turns on debug mode for debugging queries. Specifying this argument causes the SQL statement actually submitted to the data source and the number of records returned from the query to be output.

Remarks If the SQL argument of the webDbQuery becomes too complex, or too long to fit on a single line, you may split it across multiple lines. In general, it is considered a good programming style to format your SQL statements structurally into several lines, because it makes your code much more readable. (Please refer to the following example)

See Also **webDbQueryMore**

Example

```
<webDbQuery Name="Employee List" DataSource="Employee DB"
SQL="SELECT employee_id, last_name, first_name, title, phone,
     FROM t_employee
     WHERE status=1"
MaxRows=100 TimeOut=1000 Debug >
```

webDbQueryMore Statement

Description An optional sub-tag of **webDbQuery**, which specifies additional SQL clauses to be appended to the main SQL statement.

Syntax

```
<webDbQuery Name="Query Name" DataSource="ODBC DataSource Name"
SQL="SQL Statement" [MaxRows=MaximunNumberOfRows]
[TimeOut=Milliseconds] [Debug] >
…
<webDbQueryMore SQL="Additional SQL clauses" >
```

Argument	Required	Description
SQL	Yes	Additional SQL clauses to be appended to the main SQL statement.

Remarks None.

See Also **webDbQuery**

Example

```
<webDbQuery Name="SiteSearch" DataSource="Sites Database"
  SQL=" SELECT * FROM SITES WHERE SiteType = #SiteType# " >
    <webIf #Form.City != ""#>
     <webDbQueryMore SQL=" AND City = '#Form.City#' ">
    </webIf >
    <webIf #Form.SortOrder != ""#>
     <webDbQueryMore SQL=" ORDER BY #Form.SortOrder# ">
    </webIf >
</webDbQuery>
```

webDbTransaction Statement

Description Provides database transaction processing. All queries contained within a webDbTransaction tag are treated as a transactional unit. This means that changes made to the database are not permanently committed until all queries in the transaction block have executed successfully. If an error occurs in one of the queries, all changes made by previous queries within the transaction block are rolled back. An optional sub-tag of **webDbQuery**.

Syntax

```
<webDbTransaction [Isolation="IsolationSetting"]>
…
</webDbTransaction>
```

Argument	Required	Description
Isolation	No	The ISOLATION attribute provides advanced control over how the database engine does locking during the transaction. Setting ISOLATION levels is an advanced technique which is supported by very few ODBC drivers. Consult your driver's documentation for more detailed information on the isolation levels it supports and on the behavior of the driver for each level. The valid values for the ISOLATION attribute are: • Read_Uncommitted • Read_Ccmmitted • Repeatable_Read • Serializable • Versioning

Remarks Not every driver supports transactions (e.g., the SQL Server and Access drivers do while the FoxPro, dBase, and Paradox drivers do not), and not every driver that supports transactions supports all ISOLATION levels (e.g., the Access driver supports only READ_COMMITTED). When you attempt to use a transaction/isolation-level combination for a specific driver, Web+™ queries the driver for its transaction capabilities and returns an error if the driver indicates that it is not capable of implementing the request.

See Also **webDbQuery, webDbRollBack**

Example

```
<webDbTransaction>
    <webDbQuery Name="CashWithdraw" DataSource="Checking
Account"
        SQL = "UPDATE Accounts SET Balance = Balance - #Amount#
                WHERE Account_ID = #AccountFrom# ">
```

```
    <webDbQuery Name="CashDeposit" DataSource="Checking Account"
        SQL = "UPDATE Accounts SET Balance = Balance + #Amount#
               WHERE Account_ID = #AccountTo# ">
</webDbTransaction>
```

webDbRollBack Statement

Description Undo all the changes made from <**webDbTransaction**>.

Syntax
```
<webDbRollBack>
```

Remarks

See Also **webDbQueryMore, webDbTransaction**

Example
```
<webDbTransaction>
    <webDbQuery Name="CashWithdraw" DataSource="Checking
Account"
        SQL = "UPDATE Accounts SET Balance = Balance - #Amount#
               WHERE Account_ID = #AccountFrom# ">
    <webDbQuery Name="CashDeposit" DataSource="Checking Account"
        SQL = "UPDATE Accounts SET Balance = Balance + #Amount#
               WHERE Account_ID = #AccountTo# ">
</webDbTransaction>


<!-- Undo all the changes from the above <webDbTransaction> -->
<webDbRollBack>
```

webPrint Statement

Description Prints information based on the records returned by a query.

Syntax
```
<webPrint [Query=QueryName] [File=FileName]
[MaxRows=MaximunNumberOfRows] [Group=GroupByName] >
…
</webPrint>
```

Argument	Required	Description
Query	Yes	The name of the query as specified in <**webDbQuery**> from which you want to draw data for the output section.
MaxRows	No	The maximum number of record rows to be displayed in the output section.
Group	No	The name(s) of the database column(s) with which output will be grouped by.

Remarks When using the **Group** attribute with **webPrint** you must sort the results of your query by the field which is used for the grouping. This is illustrated in the following example, where the **Group** field (CourseLevel) is also specified in the ORDER BY clause of the SQL statement.
There is no limit to the number of **webPrint** statements which may be

nested together. If you wish to use multiple levels of grouping, you also need to have multiple levels of sorting in your SQL query (e.g., "ORDER BY Region, State").

See Also **webDbQueryMore**

Example

```
<webDbQuery Name="EmployeeList" DataSource="Employee DB"
  SQL="SELECT *
         FROM t_employee
         WHERE DepartmentID = '#Form.Department#'
         ORDER BY StateID ">


<webIf #EmployeeList.RecordCount = 0#>
    There is no employees in this department.
<webElse>
    <webPrint Query="EmployeeList" Group="DepartmentID" >
    <P> <H4>#EmployeeList.State#</H4>
    <UL>
       <webPrint>
          <LI> #StateID#  #StateName#
       </webPrint>
    </UL>
    </webPrint>
</webIf>
```

webPrintTable Statement

Description Printing query results in tabular format.

Syntax

```
<webPrintTable [Query=QueryName] [File=FileName
Delimiter=DelimiterString ] [MaxRows=MaximunNumberOfRows]
[ColSpacing=n] [ColSpacing=n] [ColHeaders] [HeaderLines=n]
[HTMLTable [WidthStyle={"percent"|"pixel"}]] >
…
<webPrintTableCol Header=HeaderName [Width=n] [Align="{Left|
Right|Center}"] [Text=Text] >
…
</webPrintTable>
```

Argument	Required	Description
Query	Yes	The name of the query as specified in <**webDbQuery**> from which you want to draw data for the output section.
MaxRows	No	The maximum number of record rows to be displayed in the output section.
ColSpacing	No	The number of spaces between columns. Default Value = 2
ColHeaders	No	Specifies whether column headers are displayed above each column. If this tag is present, column headers are printed above each column. The contents of the headers are controlled by the **Header** argument of the **webPrintTableCol** tag.
HeaderLines	No	Specifies the number of lines to use for the table header. Default Value = 2 (this leaves one line between the headers and the first row of the table.)
HTMLTable	No	Displays the table as an HTML 3.0 table format
WidthStyle	No	Specifies the HTML table Width style, either as percent or as

		fixed pixel. Possible values: Percent (Default value) Pixel

Remarks When using the **Group** attribute with **webPrint** you must sort the results of your query by the field which is used for the grouping. This is illustrated in the following example, where the **Group** field (CourseLevel) is also specified in the ORDER BY clause of the SQL statement.
There is no limit to the number of **webPrint** statements which may be nested together. If you wish to use multiple levels of grouping, you also need to have multiple levels of sorting in your SQL query (e.g., "ORDER BY Region, State").

See Also **webDbQuery, webPrintTableCol**

Example

```
<webDbQuery Name="Customer DB" DataSource="Customers"
  SQL="SELECT *
         FROM customerTable
         WHERE CustomerID = '#Form.ID#'
         ORDER BY LastName ">

<webPrint Query="Customer DB" Group="CustomerID" >
<P> <H4>#State#</H4>
<UL>
<webPrint>
<LI> #StateID#  #StateName#
</webPrint>
</UL>
</webPrint>
```

webPrintTableCol Statement

Description Prints information from a query in tabular format.

Syntax

```
<webPrintTable [Query=QueryName] [File=FileName
Delimiter=DelimiterString] [MaxRows=MaximunNumberOfRows]
[ColSpacing=n] [ColSpacing=n] [ColHeaders] [HeaderLines=n]
[HTMLTable] >
…
<webPrintTableCol Header=HeaderName [Width=n] [Align="{Left|
Right|Center}"] [Text=Text] >
…
</webPrintTable>
```

Argument	Required	Description
Header	Yes	The text to use for the column's header.
Width	No	The width of the column in characters. If the length of the data to be displayed is longer than this, it will be truncated to fit. Default Value = 20
Align	No	Specifies column alignment. Possible values: **Left**, **Right**, or **Center** Default Value=Left
Text	No	Specifies the display format for the output column text according to the HTML tags and dynamic parameters contained in the Text

		attribute. Double-quote (") is used to delimit text which determines what will be displayed in the column. It can consist of a combination of literal text, HTML tags, and query result set field references. This means you can embed hyperlinks, image references, and even input widgets within table columns. Remarks Since double-quotes (") and pound signs (#) are used as the delimiters for the **Text** attribute and as the field delimiter within the **Text** attribute, respectively, both require special syntax for inclusion in the **Text** attribute. To specify a single double-quote, use two double-quotes (""). To specify a single pound sign, use two pound signs (##).

Remarks When using the **Group** attribute with **webPrint** you must sort the results of your query by the field which is used for the grouping. This is illustrated in the following example, where the **Group** field (CourseLevel) is also specified in the ORDER BY clause of the SQL statement.
There is no limit to the number of **webPrint** statements which may be nested together. If you wish to use multiple levels of grouping, you also need to have multiple levels of sorting in your SQL query (e.g., "ORDER BY Region, State").

See Also **webDbQuery, webPrintTable**

Example

```
<WebPrintTable Query="Customer DB" HTMLTable border>
    <WebPrintTableCol Header="First Name" Width=25 Align=Left
    Text=#FirstName#>
    <WebPrintTableCol Header="Last Name" Width=25 Align=Left
    Text=#LastName#>
    <WebPrintTableCol Header="Work Phone" Width=20 Align=Center
    Text=#WorkPhone#>
</WebPrintTable>
```

Cookie Statements

webCookie Statement

Description Stores a cookie onto the browser.

Syntax

```
<webCookie Name=CookieVariableName Value=CookieValue
Expires=Expiration [Secure=securityoption]>
```

Argument	Required	Description
Name	Yes	The name of the cookie variable.
Value	Yes	The value to be assigned to the cookie variable.
Expires	Yes	The expiration of the cookie variable. Can be specified as a date (e.g. '12/31/96'), number of days (e.g. 10, 100), NOW or NEVER. Using NOW effectively deletes the cookie from the client's browser.
Secure	No	Specifies the security requirement for transmitting the cookie variable. If the browser does not support SSL security then the cookie will not be sent.

Remarks

See Also

Example This example stores the user's ID number 12345 as a variable ('UserID') in his or her browser. The cookie variable will expire in 100 days.

```
<webCookie Name="UserID" Value="12345" Expires="100">
```

The following shows how to display the 'UserID' cookie variable stored in the above example:

```
<webPrint>
  #Cookie.UserID#
</webPrint>
```

The following show how to test the value of the variable within a webIf statement:

```
<webIf #Cookie.UserID = "12345"#>
```

TCP/IP Communication Statements

webSocketOpen Statement

Description Opens a socket for TCP/IP communication with another host.

Syntax

```
<webSocketOpen DomainName=DomainName|IPAddress=IPAddress
PortNumber=PortNumber Handle=#HandleVar# RetVar=#ReturnValue#
[TimeOut=#n#]>
```

Argument	Required	Description
DomainName or IPAddress	Yes	The domain name or standard 32-bit IP address of the host to which you want to connect.
PortNumber	Yes	The protocol port of the remote machine.
Handle	Yes	A variable in which the handle of the new socket is stored.
RetVar	Yes	A variable which will be set to 1 if the socket is successfully opened or 0 if it is not.
TimeOut	No	The maximum time Web+ will wait for the connection to be made before returning an error message. By default, it will use the time under the SocketTimeOut keyname in the file WEBPSVR.INI..

Remarks A socket is the interface by which a program communicates with another host machine. Before your Web+ scripts can communicate with another computer, you must open a socket to that computer. Then you may communicate with it by sending and receiving text.

A socket handle is an ID number used to identify a socket. Whenever you want to send or receive text using a socket, or close a socket, you must supply the handle for the socket you want to use.

Protocol Ports are numbers associated with various protocols used for transferring information from one host to another. By supplying a port number you can tell the remote machine what protocol you intend to use. The way a remote machine responds to your communication depends on what communication protocol you use.

Here are some commonly used protocol port numbers:

Protocol	Port #
Echo Protocol	7
File Transfer Protocol (FTP)	21
Telnet Protocol	23
Simple Mail Transfer Protocol (SMTP)	25

See Also <**webSocketClose**>, <**webSocketSendText**>, <**webSocketRcvText**>

Example

<!-- First, use webSocketOpen to open a socket to search.yahoo.com using the HTTP port (80). Second, use webSocketSendText to send a message to request a search on the string "TalentSoft". Then, use webSocketRcvText to receive search result from yahoo and store the result in the Text variable. Afterwards, close socket using webSocketClose. Last, display the search result stored in the Text variable. --->

```
<webSocketOpen DomainName="search.yahoo.com" PortNumber=80
Handle=#Sock# RetVar=#Ret#>
<!-- If the connection was successful, ... -->
<webIf #Ret#>
    <!--- Send some text to yahoo --->
    <webSocketSendText Handle=#Sock#  Text=#"GET /bin/search?p=talentsoft" & Chr(13) &
Chr(10)# RetVar=#Ret#>
    <!--- If the text was sent successfully, ... --->
    <webIf #Ret#>
        <!--- Receive some text from yahoo and print it out --->
        <webSet #Tag# = 1>
        <webWhile #Tag#>
            <webSocketRcvText Handle=#Sock# Text=#Text#>
            <webPrint><p>#Text#</p></webPrint>
            <webIf #InStr(Text, "</html>")#>
                <webSet #Tag# = 0>
            </webIf>
        </webWhile>
    </webIf>
    <!--- Close the socket --->
    <webSocketClose Handle=#Sock#>
</webIf>
```

WebSocketClose Statement

Description Closes a socket opened with <webSocketOpen>.

Syntax <**webSocketClose** Handle=#*Handle*# RetVar=#*ReturnValue*#>

Argument	Required	Description
Handle	Yes	The socket handle which was returned by the <webSocketOpen> statement.
RetVar	No	A variable which will be set to 1 if the socket is successfully closed or 0 if it is not.

See Also <**webSocketOpen**>, <**webSocketSendText**>, <**webSocketRcvText**>

Example

```
<webSocketClose Handle=#Sock#>
```

webSocketSendText Statement

Description Sends data through a socket opened with <webSocketOpen>.

Syntax
```
<webSocketSendText Handle=#Handle# Text=#Text#
RetVar=#ReturnValue#>
```

Argument	Required	Description
Text	Yes	The text to be sent to the remote machine.
Handle	Yes	The socket handle which was returned by the <webSocketOpen> statement.
RetVar	No	A variable which will be set to 1 if the text is successfully sent or 0 if it is not.

Remarks The amount of text which can be sent at one time is limited to about 8K. To send more, you can use multiple <webSocketSendText> statements..

See Also <**webSocketOpen**>, <**webSocketClose**>, <**webSocketRcvText**>

Example
```
<webSocketSendText Handle=#Sock# Text=#"GET /index.htm" &
Chr(13) & Chr(10)# RetVar=#Ret#>
```

webSocketRcvText Statement

Description Receives data through a socket opened with <webSocketOpen>.

Syntax
```
<webSocketRcvText Handle=#Var# Text=#Buffer#
RetVar=#ReturnValue#>
```

Argument	Required	Description
Text	Yes	A variable in which the received text will be stored.
Handle	Yes	The socket handle which was returned by the <webSocketOpen> statement.
RetVar	No	A variable which will be set to 1 if the text is successfully received or 0 if it is not.

Remarks The amount of text which can be received at one time is limited to about 8K. To send more, you can use multiple <webSocketRcvText> statements..

See Also <**webSocketOpen**>, <**webSocketClose**>, <**webSocketSendText**>

Example
```
<webSocketRcvText Handle=#Sock# Text=#Text#>
```

Other Misc. Statements

webBeep Statement

Description Sounds the bell of the Web+™ server.

Syntax <**webBeep**>

Remarks The beep sound your Web+™ server makes is a function of your hardware.

Example <**webBeep**> <**webBeep**> <**webBeep**> <**webBeep**> <**webBeep**>
The Web server is paging the system operator. Web+ just beeped five times.

webVarExistCheck Statement

Description Check for the variable exists at the prompt.

Syntax <**webVarExistCheck On>, <webVarExistCheck Off>**

Remarks If you refer a variable in any expression other than **ParameterExists** and the variable has not been assigned any value, webplus will prompt that this variable not defined. So it is better to check those variables especially from the form by using ParameterExists first and then use it.

webVarExistCheck turns on and turn off this prompt. By default, web+ assumes a On for Var Exist Check at the prompt. So you can add a line like: **<webVarExistCheck Off>** in those templates you do not like a prompt of Var Non Exist prompt.

See Also **ParameterExists** function

Example **<webVarExistCheck Off>**

Form Field Validation Suffixes

Field Suffix	Value Attribute	Explanation
_required	Custom error message.	Verifies that some entries are required by the user.
_integer	Custom error message.	Verifies that a number is entered, if the user enters a floating point value it is rounded to an integer.
_float	Custom error message.	Verifies that a number is entered, if the users entersa floating point value, it does not round it up.
_range	MIN=MinValue MAX=MaxValue	Verifies that the numeric value entered is within the specified boundaries. One or both of the boundaries (MIN and MAX) may be specified.
_date	Custom error message.	Verifies that a date of one of the following formats is entered by the user: MM/DD/YY, MM/DD/YYYY, MM/DD (assumes current year). Dash (-) separated dates are also supported.
_mask=maskstring	Custom error message	Maskstring
		0 Digit placeholder. One digit only ? Digit placeholder. Zero or more digit. . Decimal placeholder. , Thousand placeholder : Time separator / Date \ Treat the next character in the mask string as a literal. This allows you to include the '?', '&', etc in the mask C. C Character placeholder. One character only. Valid values for placeholder. c Character placeholder. Zero or more characters. A Alphanumeric character placeholder. e.g. a-z, A-Z, 0-9. a Alphanumeric character placeholder. Zero or more alphanumeric characters. Z Letter placeholder. One letter only. e.g. a-z or A-Z z Letter placeholder. Zero or more letters. | OR separator. Either nothing, or one of the items must be selected. e.g. Y|N, M|F, A|B|C|D

Variable Manipulation

Variables

Variable Type	Explanation
Form Fields	The most common way of passing parameters to scripts is using form fields. When a user enters data in a form field, a parameter bearing the name of the form field is passed to the script.
URL Parameters	Parameters are encoded after the script name in the URL (e.g., /cgi-bin/webplus.exe) using a key=value format.
CGI Environment	Every request sent to a CGI script has several environment variables sent to it that relate to the context in which it was sent. The variables available to you depend upon the browser and server software in use for a given request.
`webSet` Variables	The `webSet` tag defines a variable that is then available for use within the rest of the script.
HTTP Cookies	HTTP Cookie variables are global variables that are available to any script interacting with the browser in which the cookie is set. They are created using the webCOOKIE tag.
Queries	As soon as a query has been run you can use its results as dynamic parameters in other queries. For example, if you create a query named 'LookupUser' which finds the ID for a user given their name you might want to use this ID in another query. To do this, use the name of the query followed by a dot and the name of the field (e.g., #LookupUser.User_ID#).

Resolving ambiguities in variable names

It is possible for several variables with the same name to be included in your scripts. For example, you may write a script which contains a `<webSet>` statement setting the value of the variable `x`. A user running your script might then submit a different value for `x` in a form. To distinguish between different variables of the same name, you can use the following prefixes with your variable names:

Use this prefix:	To return the value of:
#*QueryName*.*VariableName*#	variable returned by the given query
#Variables.*VariableName*#	variable set by a `<webSet>` statement
#Form.*VariableName*#	form variable
#URL.*VariableName*#	variable entered in URL
#Cookie.*VariableName*#	cookie variable
#CGI.*VariableName*#	CGI variable

If two or more of the above exist and you use `#VariableName#` with no prefix, it will return the value of the variable appearing higher in the above chart.

Appendixes

Appendix A - ANSI Character Set

- Represents a non-printable character.

** - ANSI value 8 is a Backspace, value 9 is a Tab, value 10 is a Linefeed, and value 13 is a Carriage Return.

0	##	1	##	2	##	3	##	4	##	5	##	6	##	7	##
8	**	9	**	10	**	11	##	12	##	13	**	14	##	15	##
16	##	17	##	18	##	19	##	20	##	21	##	22	##	23	##
24	##	25	##	26	##	27	##	28	##	29	##	30	##	31	##
32		33	!	34	"	35	#	36	$	37	%	38	&	39	'
40	(| 41 |)	42	*	43	+	44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[	92	\	93	]	94	^	95	_
96	`	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	I	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124	|	125	}	126	~	127	##
128	##	129	##	130	##	131	##	132	##	133	##	134	##	135	##
136	##	137	##	138	##	139	##	140	##	141	##	142	##	143	##
144	##	145	‘	146	’	147	“	148	”	149	•	150	–	151	—
152	˜	153	™	154	š	155	›	156	œ	157	##	158	##	159	Ÿ
160		161	¡	162	¢	163	£	164	¤	165	¥	166	¦	167	§
168	¨	169	©	170	ª	171	«	172	¬	173		174	®	175	¯
176	°	177	±	178	²	179	³	180	´	181	µ	182	¶	183	·
184	¸	185	¹	186	º	187	»	188	¼	189	½	190	¾	191	¿
192	À	193	Á	194	Â	195	Ã	196	Ä	197	Å	198	Æ	199	Ç
200	È	201	É	202	Ê	203	Ë	204	Ì	205	Í	206	Î	207	Ï
208	Ð	209	Ñ	210	Ò	211	Ó	212	Ô	213	Õ	214	Ö	215	×
216	Ø	217	Ù	218	Ú	219	Û	220	Ü	221	Ý	222	Þ	223	ß
224	à	225	á	226	â	227	ã	228	ä	229	å	230	æ	231	ç
232	è	233	é	234	ê	235	ë	236	ì	237	í	238	î	239	ï
240	ð	241	ñ	242	ò	243	ó	244	ô	245	õ	246	ö	247	÷
248	ø	249	ù	250	ú	251	û	252	ü	253	ý	254	þ	255	ÿ

Appendix B - HTML Form Tags

FORM Tag

Description The <FORM> tag starts the form. A document can consist of multiple forms, but forms cannot be nested; that is, no form can be placed inside another form.

Syntax <FORM ACTION="/*cgi-bin/webplus.exe?script=webplus.wml*"" METHOD="POST">

Argument	Required	Description
ACTION	Yes	The URL of the CGI program that will process the form information.
METHOD	Yes	METHOD specifies how the server will send the form information to the program. There are two HTTP methods used to submit the form: POST - sends the data through standard input. GET(default) - passes information through environment variables.
ENCTYPE	No	The encoding scheme for the POST data. There are two ENCTYPES are allowed: **application/x-www-form-urlencoded** (default)and **multipart/form-data**.

Remarks

Example

```
<FORM ACTION="/cgi-bin/webplus.exe?script=/webplus/webplus.wml">
METHOD="Post">
</FORM>
```

INPUT Tag

Description Most form elements are implemented using <INPUT> tag. It specifies different type of input is being requested

Syntax <INPUT TYPE="*type*" NAME="*name*" SIZE="*size*" VALUE="*value*" MAXLENGTH="maxlength">

Argument	Required	Description
TYPE	Yes	It specifies the type of input is being requested Types of elements are available: *text, password, hidden, radio, checkbox, submit* and *reset*.
NAME	Yes	The NAME attribute defines the name of the particular input element.
SIZE	No	The physical size of the input field.
MAXLENGTH	No	Defines the maximum number of characters that will be accepted by the browser. By default, there is no limit.
VALUE	No	It is used to insert a default value into the field.

Remarks

Example

```
<INPUT TYPE="text" NAME="name" VALUE="value" SIZE="size">
```

SELECT & OPTION Tags

Description The SELECT tag is used to specify a list box whereas, the OPTION tag is used to specify the elements in the box.

Syntax

```
<SELECT NAME="name">
    <OPTION VALUE="option1">option1
    <OPTION VALUE="option2">option2
</SELECT>
```

Argument	Required	Description
VALUE	No	It is used to insert a default value into the field.
SELECTED	No	It indicates the option that is initially selected.

Remarks

Example

```
<SELECT NAME="VERSION">
    <OPTION VALUE="webplus2.0">Web+2.0
    <OPTION VALUE="webplus2.1">Web+2.1
</SELECT>
```

TEXTAREA Tag

Description The TEXTAREA tag creates a scrolled text field.

Syntax

```
<TEXTAREA NAME="GuestBook" ROWS=10 COLS=40>
</TEXTAREA>
```

Argument	Required	Description
ROWS	Yes	The number of rows in the text box.
COLS	Yes	The number of columns in the text box.
NAME	Yes	The NAME attribute defines the name of the particular input element.

Example

```
<TEXTAREA NAME="GUESTBOOK" ROWS=10 COLS=40>
</TEXTAREA>
```

HTML Form Tags Summary

Form Tag	Description
<FORM ACTION="/cgi-bin/webplus.exe?script=/webplus.wml METHOD="POST">	Start the Form
<INPUT TYPE="text" NAME="name" VALUE="value" SIZE="size">	Text field
<INPUT TYPE="password" NAME="name" VALUE="value" SIZE="size">	Password field
<INPUT TYPE="hidden" NAME="name" VALUE="value" SIZE="size">	Hidden field
<INPUT TYPE="checkbox" NAME="name" VALUE="value" SIZE="size">	Checkbox
<INPUT TYPE="radio" NAME="name" VALUE="value" SIZE="size">	Radio button
<SELECT NAME="name" SIZE=1> <OPTION VALUE="value" SELECTED>Web+ 2.0 <OPTION VALUE="value">Web+ 2.1 : </SELECT>	Menu
<SELECT NAME="name" SIZE=n MULTIPLE>	Scrolled list
<TEXTAREA ROWS=yy COLS=xx NAME="name"> .... </TEXTAREA>	Multiline text fields
<INPUT TYPE="submit" NAME="name" VALUE="value" >	Submit button
<INPUT TYPE="image" SRC="/image" NAME="name" VALUE="value">	
<INPUT TYPE="reset" VALUE="value">	Reset button
</FORM>	Ends Form

Index