String Functions

String functions are the mechanism for the manipulation of strings in scripts.

Search in This Topic

On This Page

- CountSubString
- DQuote
- Format
- Index
- IsValidString
- Left
- LPad
- LPadChar
- LTrim
- LTrimChars
- Mid
- ParseURL
- Quote
- RegExMatch
- RegExReplace
- Replace
- Right
- RPad
- RPadChar
- RTrim
- RTrimChars
- Split
- SplitCSV
- StringLength
- ToLower
- ToProper
- ToUpper
- Trim
- TrimChars
- Truncate
- URLDecode
- URLEncode

Related Topics

- Formula Builder
- Formula Builder Debugging
- Formula Builder Features
- Using the Formula Builder

Last updated: Feb 04, 2020

CountSubString

Declaration

```
int CountSubString(string str, string subStr)
```

Syntax

```
CountSubString(<str>, <subStr>)
```

Required Parameters

- **`str`:** A string to be searched
- **`subStr`:** The string to search with

Description

Returns the number of times a sub-string appears in a string.

Examples

```
CountSubString("Mississippi", "iss"); // Returns 2
```

[Back to Top]

DQuote

Declaration

```
string DQuote(string str)
```

Syntax

```
DQuote(<str>)
```

Required Parameters

- **`str`:** A string to be double quoted

Description

Places a string in double quotes and returns the result. Embedded double quotes are not escaped.

Examples

```
DQuote("any str"); // Returns ""any str""

DQuote('any "text" str');
// Returns ""any "text" str"" -- note that the embedded double quotes are
not escaped
```

NOTE: The outer pair of double quotes in the above examples is the indication of a string data type and is not returned as part of the string itself.

[Back to Top]

Format

Declaration

```
string Format(string formatStr, type de)
```

Syntax

```
Format(<formatStr>, <de>)
```

Required Parameters

- **`formatStr:`** The format string to be used when formatting the string
- **`de:`** A data element; if necessary it will be converted to a string prior to being formatted

Description

Returns a string in the format specified by a format string.

The format specification is the same as used in the Date and Time Functions; see its section on format strings. It is similar to the standard C library `printf` format specification.

This function can be used to set the output format for a target. It is useful when Jitterbit's default output format for a data type (such as date or double) is not as desired. The call to the `Format()` function must be the last call in a mapping formula to have it be the mapped value.

Examples

```
Format("'%5d'", 2); // Returns "'    2'"
```

[Back to Top]

Index

Declaration

```
int Index(string str, string subStr[, int n])
```

Syntax

```
Index(<str>, <subStr>[, <n>])
```

Required Parameters

- **`str:`** A string to be searched
- **`subStr:`** The string to search with

Optional Parameters

- **`n:`** The specific instance of interest if more than one match

Description

Returns the position of a sub-string within a string.

In cases where the sub-string appears more than once in the main string, the third argument (n) is used to determine which specific instance of the sub-string is of interest.

- If n < 0, the instance counting starts from the end of the main string.
- If |n| > the maximum number of times the sub-string appears in the main string, the function returns -1.

Otherwise, the function returns the sub-string's first character's position within the main string (starting at position 0).

Special cases to consider:

- n = 0 returns the same result as n = 1
- If subStr is an empty string (""):
  - n >= 0 always returns 0
  - n < 0 always returns Length(str)

Examples

```
Index("Mississippi", "s"); // Returns 2
// When called with only two arguments
// the optional third argument defaults to 1,
// resulting in the index of the first match being returned

Index("Mississippi", "iss", 3)// Returns -1
// Searching for the 3rd instance of "iss" within Mississippi

Index("Mississippi", "si", 2); // Returns 6
// The 2nd instance of the sub-string
// "si" starts at position 6 as the
// first letter "M" starts at position 0

Index("Mississippi", "", 1); // Returns 0
Index("Mississippi", "", -1); // Returns 11
```

[Back to Top]

IsValidString

Declaration

```
bool IsValidString(string str)
```

Syntax

```
IsValidString(<str>)
```

Required Parameters

- **str:** A string to be validated

Description

Returns true if each character in a string is valid. Valid characters are ASCII codes 32 through 126 inclusive and any of linefeed (LF), carriage return (CR), or tab (TAB) characters.

Examples

```
IsValidString("An Invalid String™");
// Returns 0 (false), as the trademark symbol in this font is a code
greater than 128

IsValidString("A Valid Broken
String");
// Returns 1 (true)
```

[Back to Top]

Left

Declaration

```
string Left(string str, int n)
```

Syntax

```
Left(<str>, <n>)
```

Required Parameters

- **`str:`** A string
- **`n:`** The number of characters from the left end to be returned

Description

Returns n characters of a string, counting from the left (the beginning) of a string.

See also the `Mid()` and `Right()` functions.

> **NOTE:** If the string is a numeric string that begins with a leading zero, it must be enclosed within quotes or it will be interpreted as an integer and a different starting character index may result.

Examples

```
Left("01234567890", 4);
// Returns "0123"
```

[Back to Top]

LPad

Declaration

```
string LPad(string str, int n)
```

Syntax

```
LPad(<str>, <n>)
```

Required Parameters

- **`str:`** A string

- **n:** The number of characters from the left end to be returned

Description

Adds spaces to the left (the beginning) of a string until the string contains n characters. Strings containing n or more characters are truncated to n characters.

`LPad(str, -n)` is the same as `RPad(str, n)`. See the `RPad()` function.

Examples

```
LPad("01234567", 9);
// Returns " 01234567"
// Adds one space to the left side
// of the original string

LPad("1234567890", 9);
// Returns "123456789"
```

[Back to Top]

LPadChar

Declaration

```
string LPadChar(string str, string padChar, int n)
```

Syntax

```
LPadChar(<str>, <padChar>, <n>)
```

Required Parameters

- **str:** A string
- **padChar:** A string to be used to pad characters; if more than one character, the first character is used
- **n:** The number of characters from the left end to be returned

Description

Adds a padding character to the left (the beginning) of a string until the string contains n characters. Strings containing n or more characters are truncated to n characters.

`LPadChar(str, " ", n)` is the same as `LPad(str, n)`. See the `LPad()` function.

Examples

```
LPadChar(1234567, "0", 10);
// Returns "0001234567"
LPadChar(1234567, ".0", 10);
// Returns "...1234567"
```

[Back to Top]

LTrim

Declaration

```
string LTrim(string str)
```

Syntax

```
LTrim(<str>)
```

Required Parameters

- **`str:`** A string

Description

Removes whitespace (spaces, tabs) from the left (the beginning) of a string and returns the remaining characters.

Examples

```
LTrim(" Hello World!  ");
// Returns "Hello World!  "

LTrim("        Hello World!");
// Removes the leading tab character
// and returns "Hello World!"
```

[Back to Top]

LTrimChars

Declaration

```
string LTrimChars(string str, string trimChars)
```

Syntax

```
LTrimChars(<str>, <trimChars>)
```

Required Parameters

- **`str:`** A string to be trimmed from the left (the beginning)
- **`trimChars:`** A string of trimming characters to be matched; the order does not matter

Description

Removes any leading characters in a string from the left (the beginning) that match those in the trimming characters and returns the remaining characters.

This function tests each leading character of a string, beginning on the left edge, and sees if it is found in the trim characters. If it does, it is removed, and the process repeated until there is no longer a match.

This can be used to trim characters other than the default whitespace characters, such as trimming leading colons.

See also the `RTrimChars()` and `TrimChars()` functions.

Examples

```
LTrimChars("::StartMsg :: Hello :: EndMsg::", ":");
// Returns "StartMsg :: Hello :: EndMsg::"

LTrimChars("// This is a comment", " /");
// Returns "This is a comment"

LTrimChars("// This is a comment", "/ ");
// Returns "This is a comment"

LTrimChars("// This is a comment", "/");
// Returns " This is a comment"

LTrimChars("// This is a comment", " ");
// Returns "// This is a comment"
// Returns the string unchanged as a space
// was not a leading character
```

[Back to Top]

Mid

Declaration

```
string Mid(string str, int m, int n)
```

Syntax

```
Mid(<str>, <m>, <n>)
```

Required Parameters

- **`str:`** A string from which to extract a sub-string
- **`m:`** The zero-based starting character position to begin extracting the sub-string
- **`n:`** The length (number of characters) of the sub-string to be extracted

Description

Returns a portion of a string, starting with the character at position `m` for a length of `n` characters.

See also the `Left()` and `Right()` functions.

> **NOTE:** If the string is a numeric string that begins with a leading zero, it must be enclosed within quotes or it will be interpreted as an integer and a different starting character index may result.

Examples

```
Mid("Jitterbit", 2, 3);
// Returns "tte"

Mid("01234567890", 7, 3);
// Returns "789"

// Passing the same string as an
// integer with a leading zero
// returns a different result
Mid(01234567890, 7, 3);
// Returns "890"
```

[Back to Top]

ParseURL

Declaration

```
array ParseURL(string url)
```

Syntax

```
ParseURL(<url>)
```

Required Parameters

- **`url:`** A string containing a URL for parsing

Description

Parses a URL string and returns an array of decoded values. The values are tagged so that they can be retrieved either by index or by field name.

When retrieving values from the result, the case of a field name is ignored. See also the `URLDecode()` a nd `URLEncode()` functions.

Examples

```
url = "http://hostname/user?email=john1990%40example.com&name=John%
20Smith";
arr = ParseURL(url);

email = arr[0];
// "email" will be john1990@example.com
email = arr["email"];
// same as previous

name = arr["name"];
// "name" will be "John Smith"
name = arr["Name"];
// Same as previous;
// case is ignored for field names
```

[Back to Top]

Quote

Declaration

```
string Quote(string str)
```

Syntax

```
Quote(<str>)
```

Required Parameters

- **`str:`** A string to be single quoted

Description

Places a string in single quotes and returns the result. Embedded single quotes are not escaped.

Examples

```
Quote("Any string");
// Returns "'Any string'"

Quote("Ain't escaped");
// Returns "'Ain't escaped'" -- note that the embedded quote is not escaped
```

NOTE: The outer pair of double quotes in the above examples is the indication of a string data type and is not returned as part of the string itself.

[Back to Top]

RegExMatch

Declaration

```
int RegExMatch(string str, string exp[, type var1,... type varN])
```

Syntax

```
RegExMatch(<str>, <exp>[, <var1>,... <varN>])
```

Required Parameters

- **`str:`** A string to be single quoted
- **`exp:`** A regular expression that consists of groups

Optional Parameters

- **`var1...varN:`** One or more variable names, to be matched to the marked sub-expressions of the expression

Description

Matches a regular expression with an input string, stores the marked sub-expressions in variables, and returns the number of matches.

Returns the total number of marked sub-expressions (which could be more or less than the number of variables actually supplied). Any additional matches that exceed the number of variables supplied are discarded. If there are no matches, `-1` is returned.

The regular expression follows the Boost regular expression library syntax. It is a variation of the Perl regular expression syntax.

See also the `RegExReplace()` function.

Examples

```
result = RegExMatch("[abc]", "(\\[)(.*)(\\])", "dummy", "value");
// Sets the global variable "$dummy" to "[".
// Sets the global variable "$value" to "abc".
// Returns 3 because the character "]"
// also matched the last sub-expression.
// However, as there are only two variable names,
// the last match is not saved to a variable.
```

NOTE: "\" (backslash) has to be escaped in literal strings since it is the escape character. To use a "\", use a doubled backslash: "\\".

[Back to Top]

RegExReplace

Declaration

```
string RegExReplace(string str, string exp, string format)
```

Syntax

```
RegExReplace(<str>, <exp>, <format>)
```

Required Parameters

- **`str:`** A string to be searched and have any matching sub-strings replaced
- **`exp:`** The regular expression in the Boost regular expression library syntax to be used to match for sub-strings of the string
- **`format:`** A format string in the Boost-Extended Format String Syntax

Description

Replaces all sub-strings in a string that match an expression, using a specifed format to replace each sub-string. Any sections of the string that do not match the expression are passed through to the returned string unchanged.

The regular expression follows the Boost regular expression library syntax. It is a variation of the Perl regular expression syntax.

The format string follows the Boost-Extended Format String Syntax. If the format string is an empty string ("") then a match produces no result for that sub-string.

See also the `RegExMatch()` function.

Examples

```
// To remove any whitespace from a string:
RegExReplace(" t e st ss s", "\\s", "");
// Returns "testsss"

// To swap the order of sub-strings:
RegExReplace("abc(first)123(second)xyz",
             "(.*)(\\(.*\\))(.*)(\\(.*\\))(.*)",
             "\\1aaa\\4\\2\\3\\5");
// Returns "abcaaa(second)(first)123xyz"
```

NOTE: "\" (backslash) has to be escaped in literal strings since it is the escape character. To use a "\", use a doubled backslash: "\\".

[Back to Top]

Replace

Declaration

```
string Replace(string str, string old, string new)
```

Syntax

```
Replace(<str>, <old>, <new>)
```

Required Parameters

- **`str:`** A string to be searched and matching sub-strings replaced
- **`old:`** A string to be used for the match
- **`new:`** A string to be used as a replacement for any matches found

Description

Searches a string for sub-strings matching the "old" argument and replaces a matching sub-string with the "new" argument.

For more complex search and replace operations, see the `RegExReplace()` function.

Examples

```
Replace("Monday Tuesday", "day", "day night,");
// Returns "Monday night, Tuesday night,"
```

[Back to Top]

Right

Declaration

```
string Right(string str, int n)
```

Syntax

```
Right(<str>, <n>)
```

Required Parameters

- **`str:`** A string
- **`n:`** The number of characters from the right end to be returned

Description

Returns n characters of a string, counting from the right (the end) of a string.

See also the `Left()` and `Mid()` functions.

> ⚠ **NOTE:** If the string is a numeric string that begins with a leading zero, it must be enclosed within quotes or it will be interpreted as an integer and a different starting character index may result.

Examples

```
Right("01234567890",4);
// Returns "7890"
```

[Back to Top]

RPad

Declaration

```
string RPad(string s, int n)
```

Syntax

```
string RPad(string s, int n)
```

Required Parameters

- **`str:`** A string
- **`n:`** The number of characters from the right end to be returned

Description

Adds spaces to the right (the end) of a string until the string contains n characters. Strings containing n or more characters are truncated to n characters.

`RPad(str, -n)` is the same as `LPad(str, n)`. See the `LPad()` function.

Examples

```
RPad("01234567",9);
// Returns "01234567 "
```

[Back to Top]

RPadChar

Declaration

```
string RPadChar(string str, string padChar, int n)
```

Syntax

```
RPadChar(<str>, <padChar>, <n>)
```

Required Parameters

- **`str:`** A string
- **`padChar:`** A string to be used to pad characters; if more than one character, the first character is used
- **`n:`** The number of characters from the right end to be returned

Description

Adds a padding character to the right (the end) of a string until the string contains n characters. Strings containing n or more characters are truncated to n characters.

`RPadChar(str, " ", n)` is the same as `RPad(str, n)`. See the `RPad()` function.

Examples

```
RPadChar(1234567, "0", 10);
// Returns "1234567000"
```

[Back to Top]

RTrim

Declaration

```
string RTrim(string str)
```

Syntax

```
RTrim(<str>)
```

Required Parameters

- **`str:`** A string

Description

Removes whitespace (spaces, tabs) from the right (the end) of a string and returns the remaining characters.

Examples

```
RTrim(" Hello World!  ");
// Returns " Hello World!"

RTrim("Hello World!        ");
// Removes the trailing tab character
// and returns "Hello World!"
```

[Back to Top]

RTrimChars

Declaration

```
string RTrimChars(string str, string trimChars)
```

Syntax

```
RTrimChars(<str>, <trimChars>)
```

Required Parameters

- **`str:`** A string, to be trimmed from the right (the end)
- **`trimChars:`** A string of trimming characters to be matched; the order does not matter

Description

Removes any trailing characters in a string from the end that match those in the trimming characters and returns the remaining characters.

This function tests each trailing character of a string, beginning on the right edge, and sees if it is found in the trim characters. If it does, it is removed, and the process repeated until there is no longer a match.

This can be used to trim characters other than the default whitespace characters, such as trimming trailing colons.

See also the `LTrimChars()` and `TrimChars()` functions.

Examples

```
RTrimChars("::StartMsg :: Hello :: EndMsg::", ":");
// Returns "::StartMsg :: Hello :: EndMsg"

RTrimChars("// This is a comment //", " /");
// Returns "// This is a comment"

RTrimChars("// This is a comment //", "/");
// Returns "// This is a comment "

RTrimChars("// This is a comment //", " ");
// Returns "// This is a comment //"
// Returns the string unchanged as a space
// was not a trailing character
```

[Back to Top]

Split

Declaration

```
array Split(string str, string delimiter)
```

Syntax

```
Split(<str>, <delimiter>)
```

Required Parameters

- **`str:`** A string to be split
- **`delimiter:`** A string of to be matched as the split between each array element

Description

Splits a string using a delimiter string, returning the result as an array.

Returns the result in an array of size equal to the number of delimiters plus one. If the expression ends with the delimiter, the array size is equal to the number of delimiters (the last empty value is ignored). If the string contains no delimiters, an array of size one is returned containing the original string.

Examples

```
arr = Split("Donald,Minnie,Goofy", ",");
donald = arr[0];
minnie = arr[1];
goofy = arr[2];

Split("Donald.,Minnie.,Goofy.,", ".,");
// Returns the array {Donald, Minnie, Goofy}
```

This example converts an IP address (`"10.6.10.1"`) to a single integer (`168167937`):

```
// Logic: (first octet * 16777216) + (second octet * 65536) +
//   (third octet * 256) + (fourth octet)
ip = "10.6.10.1";
ipList = Split(ip, ".");
(Int(ipList[0]) * 16777216) + (Int(ipList[1]) * 65536) +
    (Int(ipList[2]) * 256) + (Int(ipList[3]));
```

[Back to Top]

SplitCSV

Declaration

```
array Split(string str[, string delimiter, int qualifier])
```

Syntax

```
Split(<str>[, <delimiter>, <qualifier>])
```

Required Parameters

- **`str:`** A string to be split

Optional Parameters

- **`delimiter:`** A string of to be matched as the split between each array element
- **`qualifier:`** A string of to be matched as the split between each array element

Description

Splits a CSV-formatted string and returns an array with the individual column values.

By default, the delimiter is a comma (,) and the string qualifier is a double quote ("). This can be changed by specifying the optional second and third arguments respectively.

NOTE: The optional parameters (delimiter and qualifier) can be specified with an integer corresponding to the ASCII code of the delimiter or qualifier character.

Examples

```
arr = SplitCSV('"Donald","Minnie ""The Mouse""", Goofy');
donald = arr[0];
// Returns "Donald"
minnie = arr[1];
// Returns "Minnie ""The Mouse"""
goofy = arr[2];
// Returns "Goofy"
```

[Back to Top]

StringLength

Declaration

```
int StringLength(string str)

array StringLength(array arr)
```

Syntax

```
StringLength(string str)

StringLength(array arr)
```

Required Parameters

- **`str:`** (First form) A string

- **arr:** (Second form) An array

Description

Returns the length of a string.

The function returns an array if the argument is an array, with each element of the returned array the length of the corresponding element of the argument array. The `Length()` function returns the length of an array rather than the length of the individual elements.

Examples

```
arr = {a="abcd", b="e"};

StringLength(arr);
// Returns the array {4,1}

Length(arr);
// Returns 2
```

[Back to Top]

ToLower

Declaration

```
string ToLower(string str)
```

Syntax

```
ToLower(<str>)
```

Required Parameters

- **str:** A string

Description

Converts all ASCII uppercase characters (A through Z, ASCII 65 through 90) in a string to lowercase.

Examples

```
ToLower("Hello World!");
// Returns "hello world!"
```

[Back to Top]

ToProper

Declaration

```
string ToProper(string str)
```

Syntax

```
ToProper(<str>)
```

Required Parameters

- **`str:`** A string

Description

Converts a string to proper case (the first letter of every word being capitalized). This is distinct from title case, which only capitalizes selected and longer words in a string.

WARNING: The `ToProper()` function is such that names such as `"mccartney"` beginning with `"mc"` are only converted correctly to `"McCartney"` if the word is preceded by a space character. To have a string that begins with `"mc"` such as `"mccartney, paul"` converted correctly to `"McCartney, Paul"`, use code such as this to add a preceding space and then immediately remove it:

```
s = "mccartney, paul";
s = Right( ToProper( " " + s), Length(s));
```

Examples

```
ToProper("bob dylan");
// Returns "Bob Dylan"
ToProper("arthur c. Clarke");
// Returns "Arthur C. Clarke"
ToProper("Peter o'toole");
// Returns "Peter O'Toole"
```

[Back to Top]

ToUpper

Declaration

```
string ToUpper(string <str>)
```

Syntax

```
ToUpper(<str>)
```

Required Parameters

- **`str:`** A string

Description

Converts all ASCII lowercase characters (a through z, ASCII 97 through 122) in a string to uppercase.

Examples

```
ToUpper("Hello World!");
// Returns "HELLO WORLD!"
```

[Back to Top]

Trim

Declaration

```
string Trim(string str)
```

Syntax

```
Trim(<str>)
```

Required Parameters

- **`str:`** A string

Description

Removes whitespace from the beginning and end of a string and returns the remaining characters.

Examples

```
Trim(" Hello World! ");
// Returns "Hello World!"
```

[Back to Top]

TrimChars

Declaration

```
string TrimChars(string str, string trimChars)
```

Syntax

```
TrimChars(<str>, <trimChars>)
```

Required Parameters

- **`str:`** A string to be trimmed
- **`trimChars:`** A string of trimming characters to be matched; the order does not matter

Description

Removes any leading or trailing characters in a string that match those in the trimming characters and returns the remaining characters.

This function tests each leading and trailing character of a string and sees if it is found in the trim characters. If it does, it is removed, and the process repeated until there are no longer any matches.

This can be used to trim characters other than the default whitespace characters, such as trimming colons.

See also the `LTrimChars()` and `RTrimChars()` functions.

Examples

```
TrimChars("::StartMsg :: Hello :: EndMsg::", ":");
// Returns "StartMsg :: Hello :: EndMsg"

TrimChars("/* This is a comment */", " /*");
// Returns "This is a comment"
```

[Back to Top]

Truncate

Declaration

```
string Truncate(string str, int firstChars, int lastChars)
```

Syntax

```
Truncate(<str>, <firstChars>, <lastChars>)
```

Required Parameters

- **`str`:** A string
- **`firstChars`:** The number of characters to delete from the left (the beginning) of the string
- **`lastChars`:** The number of characters to delete from the right (the end) of the string

Description

Deletes `firstChar` characters from the left (the beginning) and `lastChars` characters from the right (the end) of a string and returns the remaining string.

See also the `Left()` and `Right()` functions.

Examples

```
Truncate("a,b,c,", 2, 1);
// Returns "b,c"
```

[Back to Top]

URLDecode

Declaration

```
string URLDecode(string url, string paramName)
```

Syntax

```
URLDecode(<url>, <paramName>)
```

Required Parameters

- **`url`:** A string containing a URL
- **`paramName`:** The name of a parameter in the URL whose value is to be returned

Description

Parse a URL string and return the decoded value of the URL parameter with the specified name. The case of the name is ignored. If the name is not found in parameters of the URL, an empty string ("") is returned.

See also the `ParseURL()` and `URLEncode()` functions.

Examples

```
url = "http://hostname/user?email=john1990%40gmail.com&name=John%20Smith";
URLDecode(url, "name");
// Returns "John Smith"
```

[Back to Top]

URLEncode

Declaration

```
string URLEncode(string str[, int encodingOption])
```

Syntax

```
URLEncode(<str>[, <encodingOption>])
```

Required Parameters

- **`url:`** A string containing a value to encoded following the rules of RFC 1738

Optional Parameters

- **`encodingOption:`** An integer (0 - 3) that specifies the encoding to be used. Default value is 0. See the list below.

Description

Encodes a string following RFC 1738.

Valid values for the optional encoding option are:

- **`0:`** Standard URL encoding. Encodes ASCII control characters, non-ASCII characters, "reserved" characters, and "unsafe" characters. This is the default encoding if the option is omitted.
- **`1:`** Don't encode "unsafe" characters: " < ;> ; # % { } | \ ^ ~ [ ] ` and the space character
- **`2:`** Don't encode "reserved" characters: ; / ? : @ & =
- **`3:`** Don't encode "unsafe" characters and "reserved" characters

These characters are considered "safe" and are never encoded: $ - _ . + ! * ' ( ) ,

See also the `ParseURL()` and `URLDecode()` functions.

Examples

```
value1 = "FIRST#"; value2 = "<LAST>";
url = "http://hostname/get_doc?" +
    "name1=" + URLEncode(value1) +
    "&name2=" + URLEncode(value2);
// Returns
// http://hostname/get_doc?name1=FIRST%23&name2=%3CLAST%3E
```

[Back to Top]