Text Validation Functions

This function provides for a validation script the same validation as can be defined for a text document file format.

Search in This Topic

Related Topics

- Cloud Studio
- Functions
- Scripts

Last updated: Mar 27, 2019

Validate

Declaration

```
bool Validate(string op, string arg[, boolean ignoreCase])
```

Syntax

```
Validate(<op>, <arg>[, <ignoreCase>])
```

Required Parameters

- **`op:`** The comparison operator to be used for the validation
- **`arg:`** The value to be used with the comparison operator in the validation

Optional Parameters

- **`ignoreCase:`** Whether to ignore case in performing the validation; default, `false`

Description

This function is intended to be called from a script used for text validation. It provides for that validation script the same validation as can be defined for a text document file format. Typically this function is called in a script used in the validation of a source text document file format. Note that validation works only for sources, not targets.

The `Validate()` function assumes that the calling script has these three input arguments provided to it, as it uses them to obtain the value that it is going to validate:

- **`_1:`** The input value, as a string
- **`_2:`** A string with the data type of the input value (the "Type" field as used in the File Format)
- **`_3:`** A format string of the input value (the "Format" field as used in the File Format)

See the `ArgumentList()` function for more information on how values are passed using the "*_n*" syntax.

Comparison Operators and Values

Depending on the type of data passed to the called script, different comparison operators and values can be used. In all cases, a regular expression can be used that follows the Boost regular expression library syntax. It is a variation of the Perl regular expression syntax.

Data Type	Comparison Operators	Comparison Values

String	`=`
	`!=`
	`Contains`
	`Does Not Contain`
	`One Of`
	`Not One Of`
	`Regex`
		String values

Number (Integer, Long, Float, Double)	`<`
	`<=`
	`>`
	`>=`
	`=`
	`!=`
	`Between`
	`Not Between`
	`Regex`
		Integer, Long, Float, or Double values

Date	`<`
	`<=`
	`>`
	`>=`
	`=`
	`!=`
	`Between`
	`Not Between`
	`Regex`
		Dates (or strings that represent a date) using the Date Comparison Values

Boolean	`=`
	`!=`
	`Regex`
		`true, false, 1, 0`

Date Comparison Values

For dates, these comparison values can be used, in addition to dates themselves. Note that some variations use an integer parameter to indicate a quantity.

Comparison Value	Description
`YESTERDAY`	Starts 12:00:00 the day before and continues for 24 hours.
`TODAY`	Starts 12:00:00 of the current day and continues for 24 hours.
`TOMORROW`	Starts 12:00:00 after the current day and continues for 24 hours.
`LAST_WEEK`	Starts 12:00:00 on the first day of the week before the most recent first day of the week and continues for seven full days. First day of the week is determined by the current locale.
`THIS_WEEK`	Starts 12:00:00 on the most recent first day of the week before the current day and continues for seven full days. First day of the week is determined by your locale.
`NEXT_WEEK`	Starts 12:00:00 on the most recent first day of the week after the current day and continues for seven full days. First day of the week is determined by your locale.
`LAST_MONTH`	Starts 12:00:00 on the first day of the month before the current day and continues for all the days of that month.
`THIS_MONTH`	Starts 12:00:00 on the first day of the month that the current day is in and continues for all the days of that month.
`NEXT_MONTH`	Starts 12:00:00 on the first day of the month after the month that the current day is in and continues for all the days of that month.
`LAST_90_DAYS`	Starts 12:00:00 of the current day and continues for the last 90 days.
`NEXT_90_DAYS`	Starts 12:00:00 of the current day and continues for the next 90 days.
`LAST_N_DAYS:`*n*	For the number *n* provided, starts 12:00:00 of the current day and continues for the last *n* days.
`NEXT_N_DAYS:`*n*	For the number *n* provided, starts 12:00:00 of the current day and continues for the next *n* days.
`THIS_QUARTER`	Starts 12:00:00 of the current quarter and continues to the end of the current quarter.
`LAST_QUARTER`	Starts 12:00:00 of the previous quarter and continues to the end of that quarter.
`NEXT_QUARTER`	Starts 12:00:00 of the next quarter and continues to the end of that quarter.
`NEXT_N_QUARTERS:`*n*	Starts 12:00:00 of the next quarter and continues to the end of the *n*th quarter.

`LAST_N_QUARTERS:`*n*	Starts 12:00:00 of the previous quarter and continues to the end of the previous *n*th quarter.
`THIS_YEAR`	Starts 12:00:00 on January 1 of the current year and continues through the end of December 31 of the current year.
`LAST_YEAR`	Starts 12:00:00 on January 1 of the previous year and continues through the end of December 31 of that year.
`NEXT_YEAR`	Starts 12:00:00 on January 1 of the following year and continues through the end of December 31 of that year.
`NEXT_N_YEARS:`*n*	Starts 12:00:00 on January 1 of the following year and continues through the end of December 31 of the *n*th year.
`LAST_N_YEARS:`*n*	Starts 12:00:00 on January 1 of the previous year and continues through the end of December 31 of the previous *n*th year.

Examples

You can use this function in a script or mapping. First, create a script ("ValWrap") that wraps the `Validate()` function so that the local variables it requires are populated:

```
<trans>
// This code could be a script called
// "ValWrap". It would be called from
// another script or mapping.

// Argument List:
// _1 (input_string_value), _2 (data_type),
// _3 (format) _4 (comparison operator),
// _5 (comparison value), _6 (ignore case)

Validate(_4, _5, _6);
</trans>
```

Then, call the wrapper script from another script (or a mapping) to use its `Validate()` function:

```
// This code calls a wrapper script to access
// the Validate function.
// Wrapper Argument List:
// _1 (input_string_value), _2 (data_type),
// _3 (format), _4 (comparison operator),
// _5 (comparison value), _6 (ignore case)

// Validation of a Date
a1 = Date(CVTDate("02192018", "mmddyyyy", "yyyy-mm-dd HH:MM:SS"));
a2 = "Date";
a3 = ""; // Format not required if a Date passed
a4 = "=";
a5 = "THIS_QUARTER";
a6 = "true";
resultA = RunScript("<TAG>script:ValWrap</TAG>",
    a1, a2, a3, a4, a5, a6);

// Validation of a String Date
b1 = "02192018";
b2 = "Date";
b3 = "mmddyyyy";
b4 = "=";
b5 = "THIS_QUARTER";
b6 = "true";
resultB = RunScript("<TAG>script:ValWrap</TAG>",
    b1, b2, b3, b4, b5, b6);

// Validation of an Integer
c1 = 100;
c2 = "Integer";
c3 = "";
c4 = ">";
c5 = 1000;
c6 = "true";
resultC = RunScript("<TAG>script:ValWrap</TAG>",
    c1, c2, c3, c4, c5, c6);
```

[Back to Top]