MSP BUILDER

Tools for MSP Success

Multi-Tool

for Kaseya Service Desk

Reference Guide

MSP Builder
www.mspbuilder.com
Version 1.2f

MSP Builder Multi-Tool for Kaseya VSA Service Desk

MSP Builder / Baroan Technologies ("COMPANY") CONFIDENTIAL

MSP Builder
A Division of Baroan Technologies
385 Falmouth Ave
Elmwood Park, NJ 07407
201-796-0404

Contents

This page intentionally blank.

MSP Builder Multi-Tool

For Kaseya Service Desk

Overview

The MSP Builder Multi-Tool for Kaseya Service Desk is an application that runs on a Kaseya server and provides broad functionality for Service Desk procedures. The MSP Builder Multi-Tool currently provides six distinct classes of functionality – Math, Comparisons, String Manipulations, Time Calculations, Boolean logic tests, and Network calculations. A logging capability is also provided. The tool uses a standard interface with a single defined command and four arguments. The utility requires one argument – the Function ID, and accepts up to 3 additional values. These values may be required depending on the function being used. By using a standard interface to Kaseya, the MSP Builder Multi-Tool to be upgraded with future enhancements (or conversion from Demo to Licensed version) with virtually no impact to the Service Desk procedures where they are used.

Math Functions

Thirteen commonly needed functions for basic math operations. The basics of Addition, Subtraction, Multiplication, and Division are enhanced with a Mod function (returns remainder of integer division), Increment and Decrement functions to increase or decrease a value by 1, a Random Number Generator to return a random integer between 0 and 99, and conversion functions that convert a value to an Integer (decimal truncation), Rounding (to nearest decimal value), Hex/Decimal conversion, and Absolute Value to convert values to positive real numbers. All internal processing is done using real (double-precision) values to maintain high accuracy.

Comparison Functions

All six comparisons operate with double-precision numbers, eliminating the issues with Kaseya's internal comparison being string based. All values are converted to double-precision when supplied to the MSP Builder Multi-Tool. The comparisons allow Equality, Inequality, Less Than, Less Than or Equal, Greater Than, and Greater Than or Equal tests to be performed on two values.

String Manipulation Functions

A collection of sixteen functions support the most widely used string manipulation requirements. Left, Right, and Sub-String operations return part of a given string. A pair of In-String functions return either the location where one string is found in another string, or a Boolean result indicating the presence of one string in another. Other functions return the length of a string, the string in reverse character order, or replace one set of text with another in a given string. One of the most useful functions can split a delimited string and return a specific field. Other functions include ASCII conversions, Upper/Lower case conversions, and Version String comparisons.

Time Functions

Every ticket in Service Desk will likely need to deal with time in one form or another. Nineteen unique time functions start with the simple – returning a current timestamp (Date Time), Date, Time, Julian Day, Weekday name (SUN-SAT), or Weekday Number. The latter has two forms needed for most day calculations, one where Sunday is represented by 0 (zero) and the other where Sunday is represented by 7. There's a function to return a Boolean when the current day/time occurs on a weekend.

Some of the more complex functionality exists in this category, and these functions allow complex time calculations with minimal effort.

The first of these advanced functions converts between a timestamp and a modified cTime value. A cTime value represents the number of seconds from a known time epoch – usually Jan 1, 1970 or Jan 1, 1900. To keep the value reasonable, the MSP Builder Multi-Tool uses Jan 1 of the prior year as the time epoch. A common need for this is to determine what time it was or will be a specific number of seconds from (or until) now, using simple addition or subtraction of the cTime value.

The next function returns the difference between two timestamps, represented in seconds. By specifying an optional third parameter, the result can be delivered in minutes, hours, days, or years. Negative values represent the time passed since an earlier time, while positive numbers represent the "time to go" before that moment arrives.

Have you ever needed to determine the date and time of "next Tuesday" or the "Third Saturday"? The TNTO function does just that! Specify the week of the month (next, first – fourth, or last), the day of the week (next or Monday-Sunday), and the time. The function will return the timestamp that represents that time event.

Another set of functions return a Boolean value if the specified (or current) time is between two specific timestamps. This is perfect for determining if after-hours alerting procedures are needed. This can handle simple times (including spanning into the next day) or timestamps covering a time range spanning days or even years in duration.

Logic Functions

In addition to the common And, Or, and Not logic functions, an additional function evaluates various named True/False conditions, such as "True", "On", and "Yes". These all return a "1", as do any non-zero numeric values. Zero and other words return "0", signifying a "false" condition. Custom phrases that evaluate as true can be specified to expand its capability.

Network Functions

Two network-related functions are currently supported for IPV4 environments. One performs an NSLookup command and returns the host and IP Address that was resolved. Options allow returning just the hostname or IP Address. The other network function performs subnet calculations, returning a True/False value if a specific IP address is within the defined network/subnet address.

Logging

One function allows custom messages to be written to the MSP Builder Multi-Tool log file, which is maintained on the Kaseya server. This is especially useful when debugging new procedures, as messages and variable values can be written to the log even if the ticket is ultimately cancelled.

Installation

The installation package consists of just six files:

- The MSP Builder Multi-Tool Reference Guide PDF document (what you're reading now!)
- A readme.txt file with any version-specific information.
- The MSP Builder Multi-Tool Utility script – MBMTU.KIX. This script is compiled and cannot be edited or viewed.
- A second, nearly identical file called MBMTUNL.KIX is provided and replaces the MBMTU.KIX file in production to eliminate "success" messages from the log file. Only errors and log requests are written when this version is used. This is recommended in larger production environments where extremely large logs might occur.
- The SD_MBMTU.XML file, which registers the script with Kaseya.
- The Kix32.exe Kixtart administrative scripting engine. A version matched to the .KIX file is provided, but you may also download the correct version from the Kixtart web site – www.kixtart.org. If you download your own copy, please be sure to download the version specified in the readme.txt file as the MSP Builder Multi-Tool is compiled with a specific version, which may not be 100% compatible with different versions.

The installation is performed manually to allow configuration to your specific requirements. All of the tasks must be performed on your Kaseya server.

1. Start by expanding the supplied Zip file to a temp folder.
2. Move the `Kix32.exe` and `MBMTU.KIX` file to a folder of your choice. Best practice would be to create a folder for your add-on tools and utilities and place them there. For the purpose of this example, we shall assume that you place them in the `D:\Kaseya\MyTools\` folder. (If you choose to use this folder and create it, you will not need to edit the XML file in step 3.)
3. Move the SD_MBMTU.XML file to the *`d`*`:\Kaseya\xml\SDProcShellCommand\0` folder. This is the content that registers the script with Kaseya. Substitute the correct drive letter and path if your Kaseya folder is located elsewhere. You may (need to) change the last digit from zero to the partition associated with your Service Desk. The 0 folder makes the script available to all of the Service Desk instances.
    a. Edit the XML file that you just copied.
    b. Locate the "`D:\Kaseya\MyTools\`" part of the path in the CommandToExecute section and replace both instances with the correct path for your system if you are using a different location for the package.
4. *Optional* – if you want to specify a location for the log file, create the logging folder, then define a SYSTEM environment variable called "MBMTULOGPATH" and have it reference the logging folder. By default, logs are written to the "Logs" subfolder in the directory where the script is located. This folder is created automatically if it is not present.
   At this point, the Multi-Tool will operate in demo mode and only 6 functions may be used.
5. From a command prompt, navigate to the Kaseya\MyTools folder (or wherever you installed the Multi-Tool) and run "`.\kix32.exe .\mbmtu.kix --lkr`"
6. Send the License Request string via email to licensing@mspbuilder.com
   If you have paid for the Multi-Tool, you will receive a license key
   You can request a trial key that will work for 30 days if you have not yet purchased a license. Only business emails can be used to request a trial license (no @yahoo or @gmail type accounts will be accepted), and only one trial license per organization will be issued.
7. From a command prompt, navigate to the Kaseya\MyTools folder (or wherever you installed the Multi-Tool) and run "`.\kix32.exe .\mbmtu.kix --slk`". Paste the license key when prompted.

Basic Testing

You can test the MSP Builder Multi-Tool outside of Kaseya from a command prompt – just execute the following command:

```
D:\Kaseya\MyTools\Kix32 D:\Kaseya\MyTools\MBMTU.KIX --?
```

This should display the help message. Try displaying the current date and time:

```
D:\Kaseya\MyTools\Kix32 D:\Kaseya\MyTools\MBMTU.KIX TCTS
```

or adding two numbers together:

```
D:\Kaseya\MyTools\Kix32 D:\Kaseya\MyTools\MBMTU.KIX ADD 4.3 5.7
```

Be sure to specify the correct path to the scripts! If these work, you should be ready to try it from Kaseya. If they don't respond, make sure that the path to the tools is correct. The path you use here *must* match the path you defined in the SD_MBMTU.XML file.

All interactive testing should be performed from the command line run with local admin privileges.

Upgrading the MSP Builder Multi-Tool

If you receive an updated version of the MSP Builder Multi-Tool, simply overwrite the MBMTU.KIX and (if the Kix version changed) the Kix32.exe files. No other change should be required, and the update will occur with the next invocation of the MBMTU script.

Using the MSP Builder Multi-Tool in Kaseya Service Desk

Once the files are available on the Kaseya server, you are ready to utilize them from Kaseya Service Desk procedures.

It's best to start with an existing procedure, usually one that completes your ticketing process, and simply put some test code there to write the results of your BMT logic to the ticket notes. Once you're familiar with how the script process works, you can create or modify your Service Desk procedures to make use of the MSP Builder Multi-Tool. Of course, you should always test your code in a development environment before placing it into production!

In order to use the MSP Builder Multi-Tool, you need to create 4 variables using the getVariable command. The variables *must* use these names as they must match the variables defined in the SD_MBMTU.XML file.

MBMTP – this will contain the name of the Process (function) that should be performed. This value may have a prefix "NAME:" to identify the source of the request when writing events to the log file. See the Logging and Debugging section at the end of this document for more information on using the prefix.

MBMTV1-MBMTV3 – These are three variables passed to the script. All three must be defined, even when no values are passed to the script. If this is not done, Kaseya's external script processor will send an error to the MSP Builder Multi-Tool script. One word of caution – if you make many BMT script calls, always be sure to clear the MBMTV# variables if you don't need them, especially when a Process you are calling supports an optional parameter and you want the default value to be used. The optional parameters must be blank if you aren't specifically defining their values.

MBMTResult – All data is returned in this variable. It's recommended that you move the result to another named variable as each call to the MSP Builder Multi-Tool will overwrite the previous contents of the MBMTResult variable.

Basic Operation

Start by choosing the Process you want to perform. For an example, we will use the Get Current Timestamp (TCTS), which returns a string with the current date and time. No values are required, but we will define them with empty values. (Ideally, a variable called "BNULL" is defined once to contain a space character at the beginning of your code, and is used whenever a blank value is desired. This has the effect of visually indicating that the field is not defined. This notation will be used in future examples in this document.

```
getVariable("MBMTP", "Constant Value", "TCTS", "Continue on Fail")
getVariable("MBMTV1", "Constant Value", "[=BNULL=]", "Continue on Fail")
getVariable("MBMTV2", "Constant Value", "[=BNULL=]", "Continue on Fail")
getVariable("MBMTV3", "Constant Value", "[=BNULL=]", "Continue on Fail")
// Run the BMT Script
executeShellCommand("MSP BuilderMultiTool", "Continue on Fail")
// The MBMTResult variable contains the command result - save it!
getVariable("CurrentTimestamp", "Constant Value", "[=MBMTResult=]", "Continue on Fail")
```

It's that simple – the CurrentTimestamp now contains the date and time (YYYY/MM/DD hh:mm:ss) as of when the command was executed. This is a straightforward method to verify that the MSP Builder Multi-Tool is operational on your Kaseya platform.

Review the remainder of this Reference Guide to become familiar with the functions available to you. Each function is explained and examples are provided.

Note that we use "Continue on Fail" in most of our examples. This is appropriate for development and testing, but may not be the best choice for production code. Also note that our examples hard-code many of the MBMTV# values – this makes the logic obvious in the examples, but in production you would likely obtain data from other variables or custom fields.

Using a BNULL Variable

Kaseya currently does not have the ability to clear a variable once it is set. The only available choice is to use a single space character. The procedure editor always shows a null as " ", which is impossible to distinguish from a single space. We recommend that you define a variable called BNULL, define it to contain a single space, and use that anytime a blank value is needed. This will visually confirm that the variable being supplied is empty. You will see this used in the examples that follow. Of course, you may use any variable name, but "BNULL" clearly identifies it as used by the MSP Builder Multi-Tool.

Escaping Spaces

The MSP Builder Multi-Tool will ignore a single space or leading and/or trailing spaces in strings. Spaces within strings are not affected, but leading and trailing spaces will be stripped. If a single space is needed (for example, to split a string into words), use a "\s" to represent the space. Do the same if spaces must precede or follow a string of characters. Each "\s" represents a single space character.

Escaping Other Special Characters

Certain characters are difficult or impossible to specify in a Kaseya procedure, or to pass properly on the command line of a script. The following escape sequences and their corresponding special characters are supported. These escape sequences are supported only within String functions and the WLOG Process.

\^t	Tab	\^b	Backspace	\^n	CR/LF

Note that the "\^" escape mechanism was chosen as it is an unlikely combination of characters to occur in normal operations.

Error Reporting

It is possible to have an unsupported condition presented to the MSP Builder Multi-Tool. The most common is not providing the correct number of arguments, using the wrong or misspelled variable name, or forgetting to set one of the required value variables. When this occurs, the information returned will be one of the following messages:

- INVALID REQUEST - 0ARGS
  This indicates that no arguments were provided. This should not occur when called from Kaseya, but could occur when testing the process directly from a command-line. Kaseya will always pass 4 arguments based on the script definition in the XML file.
- INVALID REQUEST – PARG
  This indicates that the Process argument used to identify the function was not supplied.
- INVALID REQUEST – ARGS#
  This is the most common error, and it means that the required number of value arguments after the Process argument were not provided. Most often, the calling procedure did not define the required Kaseya variable. The "#" at the end of the "ARGS" message indicates how many value arguments were expected.
- INVALID REQUEST – PROCESS
  This message is returned when the Process specified did not match one of the available process function names. Check the spelling of the Process specified in the getVariable command when this message occurs.
- INVALID REQUEST – DEMO
  The Process requested is not available in the demo version of the MSP Builder Multi-Tool. If you require this capability, contact MSP Builder to purchase a license for the full version of the tool.

Logging

Each request made will be written to a log file. Log messages will write the date and time, the source ID (if specified), the Process requested, the optional values passed, and the result returned. If detailed logging is not required, simply overwrite the MBMTU.KIX with MBMTUNL.KIX, which suppresses the detail but still logs error messages. See the Logging and Debugging section at the end of this document for more information.

When in demo mode, the actual values that would normally be returned are written to the log file. This allows the developer to verify that the correct information is being supplied and returned. Demo mode will only return values from a limited selection of processes.

When in DEBUG mode (automatically enabled when running on a system other than the Kaseya server), only errors are written to the log file, even when full logging is enabled. Detailed results are written to the screen.

The MSP Builder Multi-Tool has built-in log management functionality that can be used to minimize the log space used. See the Log Management section later in this document for full details on this feature.

Source Identification

When many procedures are writing events to the log, it becomes difficult to identify which message represents which event. Source Identification can help reduce or eliminate any confusion.

The Process ID can be prefixed with an identifier that ends with a colon (":"). This data is stripped off to become the Source ID, leaving just the function ID to be processed. When the log is written, if the Source ID contains data, the message is modified to include "Src: <Source ID>" before the function and data values are logged. The Source ID can be any text except a colon, but should be reasonably short. Using an abbreviation of the procedure name and code block is a recommended method. For example, our ticket Intake Mapping function has 12 distinct code blocks, and each block may have several logic steps. The "INMAP_CAR03" source id represents the "Intake Mapping" procedure, "Check Allow Restricted" code block, statement group 3. With this identification in the logs, if we see an unexpected value or error message, we know exactly where to look for the issue.

Special Command-Line Arguments

There are several command line arguments available that do not return data to Kaseya. These are used to obtain or view license data or display the current version and configuration data. These arguments include:

- --KID Displays the Kaseya MSP ID associated with the license.
- --LXD Displays the current license mode, Kaseya MSP ID, the customer license key, and license status.
- --VER Displays the version of the application, the logging file path and configuration (full or limited logging), and detailed license status information.
- --LOG: Log management function – see the Log Management section at the end of this Reference Guide for detailed information.
- --LKR Generate a License Key Request string to request a paid or trial license with.
- --SLK Set the License Key to activate the complete functionality.

This page intentionally blank.

MSP Builder Multi-Tool Functions

A collection of over 60 functions to enhance the capabilities of Kaseya Service Desk procedures. Each of the available functions will be documented here, with examples for many of the functions. Each function is a three or four character mnemonic. String, Time, Logic, and Network operations are always 4 characters long and begin with “S”, “T”, “L”, or “N” respectively to identify the function group.

Math Functions

An important concept to understand is that when a program (such as Kaseya) calls another program and passes arguments, those arguments are generally passed as text strings. The MSP Builder Multi-Tool will automatically convert the variables used by the math functions into double-precision values before performing the math operation. The value is returned in a format that represents a double-precision value.

ADD - Addition

Adds two numbers together and returns the sum.

Variables

MBMTV1 The first numeric value.

MBMTV2 The second numeric value.

MBMTV3 Ignored.

Returns

A double-precision numeric value.

Example

```
getVariable(“MBMTP”, “Constant Value”, “ADD”, “Continue on Fail”)
getVariable(“MBMTV1”, “Constant Value”, “3.66”, “Continue on Fail”)
getVariable(“MBMTV2”, “Constant Value”, “4.75”, “Continue on Fail”)
getVariable(“MBMTV3”, “Constant Value”, “[=BNULL=]”, “Continue on Fail”)
// Run the BMT Script
executeShellCommand(“MSP BuilderMultiTool”, “Continue on Fail”)
// The MBMTResult variable contains the command result - save it!
getVariable(“Answer”, “Constant Value”, “[=MBMTResult=]”, “Continue on Fail”)
// The result should contain “8.41”
```

SUB - Subtraction

Subtracts V2 from V1 and returns the result.

Variables

MBMTV1 The first numeric value.

MBMTV2 The second numeric value.

MBMTV3 Ignored.

Returns

A double-precision numeric value.

Example

See ADD

MUL - Multiplication

Multiplies V1 by V2 and returns the result.

Variables

MBMTV1 The first numeric value.

MBMTV2 The second numeric value.

MBMTV3 Ignored.

Returns

A double-precision numeric value.

Example

See ADD

DIV - Division

Divides V1 by V2 and returns the result.

Variables

MBMTV1 The first numeric value.

MBMTV2 The second numeric value.

MBMTV3 Ignored.

Returns

A double-precision numeric value.

Example

See ADD

MOD – Modulo Division

Returns the remainder after dividing V1 by V2.

Variables

MBMTV1 The first numeric value.

MBMTV2 The second numeric value.

MBMTV3 Ignored.

Returns

An Integer whole number representing the remainder of a division operation.

Example

See ADD

INT – Force Integer Value

Converts V1 to an integer by truncating any decimal part.

Variables

MBMTV1 The first numeric value.

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

A double-precision numeric value with no fractional part.

Example

```
getVariable("MBMTP", "Constant Value", "INT", "Continue on Fail")
getVariable("MBMTV1", "Constant Value", "3.1524", "Continue on Fail")
getVariable("MBMTV2", "Constant Value", "[=BNULL=]", "Continue on Fail")
getVariable("MBMTV3", "Constant Value", "[=BNULL=]", "Continue on Fail")
// Run the BMT Script
executeShellCommand("MSP BuilderMultiTool", "Continue on Fail")
// The MBMTResult variable contains the command result - save it!
// The result should contain "3"
```

RND - Rounding

Rounds the V1 decimal value to a specified number of decimal places. Note that for negative values, the rounding moves to the numerically larger (more negative) value (-4.6 => -5.0 if MBMTV2 is not defined).

Variables

MBMTV1 The first numeric value.

MBMTV2 Optional – the number indicating how many places to the right of the decimal are included in the rounding. If omitted, a value with no fractional part is returned.

MBMTV3 Ignored.

Returns

A double-precision numeric value.

Example

See INT

ABS – Absolute Value

Returns the absolute value of V1. Always returns a positive value.

Variables

MBMTV1 The first numeric value.

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

A double-precision positive numeric value.

Example

See INT

INC - Increment

Increments the V1 value by 1.

Variables

MBMTV1 The first numeric value.

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

A double-precision numeric value.

Example

See INT

DEC - Decrement

Decrements the V1 value by 1.

Variables

MBMTV1 The first numeric value.

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

A double-precision numeric value.

Example

See INT

RNG – Random Number Generator

Returns a random integer between 0 and 99. The random-number generator is seeded for each invocation using a combination of uptime ms and milliseconds of the current second. This assures a fairly random result for each call.

Variables

MBMTV1 Optional – specifies the range of random numbers generated. This value defaults to 100 if not specified. Note that the minimum random number is always 0, and the maximum will be 1 less than the range specified (Min = 0; Max = Range - 1).

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

An integer numeric value.

Example

```
getVariable("MBMTP", "Constant Value", "RNG", "Continue on Fail")
getVariable("MBMTV1", "Constant Value", "[=BNULL=]", "Continue on Fail")
getVariable("MBMTV2", "Constant Value", "[=BNULL=]", "Continue on Fail")
getVariable("MBMTV3", "Constant Value", "[=BNULL=]", "Continue on Fail")
// Run the BMT Script
executeShellCommand("MSP BuilderMultiTool", "Continue on Fail")
// The MBMTResult variable contains the command result - save it!
// The result will contain an integer between 0 and 99
```

DTH – Decimal to Hex Conversion

Returns a Hex value for a supplied decimal integer.

Variables

MBMTV1 Decimal value.

MBMTV2 Optional – specify the number of digits to return. Default is to return just enough dights to represent the value.

MBMTV3 Optional – Boolean value – adds the "0x" prefix if True.

Returns

An HEX formatted string value.

Example

```
getVariable("MBMTP", "Constant Value", "DTH", "Continue on Fail") ")
getVariable("MBMTV1", "Constant Value", "4382", "Continue on Fail")
getVariable("MBMTV2", "Constant Value", "8", "Continue on Fail")
getVariable("MBMTV3", "Constant Value", "True", "Continue on Fail")
// Run the BMT Script
executeShellCommand("MSP BuilderMultiTool", "Continue on Fail")
// The MBMTResult variable contains the command result - save it!
// The result will contain "0x0000111E"
```

HTD – Hex to Decimal Conversion

Returns an integer value based on the HEX string supplied.

Variables

MBMTV1 Hexadecimal string value. The HEX prefix "0x" is supported but not required.

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

An integer numeric value.

Example

```
getVariable("MBMTP", "Constant Value", "HTD", "Continue on Fail") ")
getVariable("MBMTV1", "Constant Value", "0x0000111E", "Continue on Fail")
getVariable("MBMTV2", "Constant Value", "[=BNULL=]", "Continue on Fail")
getVariable("MBMTV3", "Constant Value", "[=BNULL=]", "Continue on Fail")
// Run the BMT Script
executeShellCommand("MSP BuilderMultiTool", "Continue on Fail")
// The MBMTResult variable contains the command result - save it!
// The result will contain "4382"
```

This page intentionally blank.

Comparison Functions

CEQ – Compare Equal

Returns TRUE (1) if V1 is Equal To V2, otherwise returns FALSE (0). Forces a mathematical comparison if both values are numbers, otherwise performs a string comparison.

Variables

MBMTV1 The first numeric or text value.

MBMTV2 The second numeric or text value.

MBMTV3 Ignored.

Returns

An Integer value in the range of 0-1.

Example

```
getVariable("MBMTP", "Constant Value", "CEQ", "Continue on Fail")
getVariable("MBMTV1", "Constant Value", "3.1524", "Continue on Fail")
getVariable("MBMTV2", "Constant Value", "3.1575", "Continue on Fail")
getVariable("MBMTV3", "Constant Value", "[=BNULL=]", "Continue on Fail")
// Run the BMT Script
executeShellCommand("MSP BuilderMultiTool", "Continue on Fail")
// The MBMTResult variable contains the command result - save it!
```

CGT – Compare Greater Than

Returns TRUE (1) if V1 is mathematically Greater Than V2, otherwise returns FALSE (0).

Variables

MBMTV1 The first numeric value.

MBMTV2 The second numeric value.

MBMTV3 Ignored.

Returns

An Integer value in the range of 0-1.

Example

See CEQ

CGE – Compare Greater Than or Equal

Returns TRUE (1) if V1 is mathematically Greater Than or Equal to V2, otherwise returns FALSE (0).

Variables

MBMTV1 The first numeric value.

MBMTV2 The second numeric value.

MBMTV3 Ignored.

Returns

An Integer value in the range of 0-1.

Example

See CEQ

CLT – Compare Less Than

Returns TRUE (1) if V1 is mathematically Less Than V2, otherwise returns FALSE (0).

Variables

MBMTV1 The first numeric value.

MBMTV2 The second numeric value.

MBMTV3 Ignored.

Returns

An Integer value in the range of 0-1.

Example

See CEQ

CLE – Compare Less Than or Equal

Returns TRUE (1) if V1 is mathematically Less Than or Equal to V2, otherwise returns FALSE (0).

Variables

MBMTV1 The first numeric value.

MBMTV2 The second numeric value.

MBMTV3 Ignored.

Returns

An Integer value in the range of 0-1.

Example

See CEQ

CNE – Compare Not Equal

Returns TRUE (1) if V1 is Not Equal to V2, otherwise returns FALSE (0). This is the inverse of CEQ. Forces a mathematical comparison if both values are numbers, otherwise performs a string comparison.

Variables

MBMTV1 The first numeric or text value.

MBMTV2 The second numeric or text value.

MBMTV3 Ignored.

Returns

An Integer value in the range of 0-1.

Example

See CEQ

NOTE:
Quantitative (LT, LE, GT, GE) comparisons are mathematical based and will return unexpected results if text string values are provided. The CEQ and CNE functions detect when both values are numeric and automatically switch between mathematical and string-based comparisons as needed.

String Manipulation Functions

SLFT – Left Part of String

Returns a subset of text from the left end of a given string.

Variables

MBMTV1 The string to evaluate.

MBMTV2 An integer value representing the left-most number of characters to return from V1.

MBMTV3 Ignored.

Returns

A string containing the selected portion of V1 text.

Example

```
getVariable("MBMTP", "Constant Value", "SLFT", "Continue on Fail")
getVariable("MBMTV1", "Constant Value", "This is a test", "Continue on Fail")
getVariable("MBMTV2", "Constant Value", "4", "Continue on Fail")
getVariable("MBMTV3", "Constant Value", "[=BNULL=]", "Continue on Fail")
// Run the BMT Script
executeShellCommand("MSP BuilderMultiTool", "Continue on Fail")
// The MBMTResult variable contains the command result - save it!
// The result should contain "This"
```

SRGT – Right Part of String

Returns a subset of text from the right end of a given string.

Variables

MBMTV1 The string to evaluate.

MBMTV2 An integer value representing the right-most number of characters to return from V1.

MBMTV3 Ignored.

Returns

A string containing the selected portion of V1 text.

Example

See SLFT

SMID – Middle Part of String

Returns a defined subset of text from the given starting position of a string.

Variables

MBMTV1 The string to evaluate.

MBMTV2 An integer value representing the starting point of the text selection.

MBMTV3 Optional - The number of characters to return beginning at the starting point. If not specified, all remaining characters between the starting point and the end of the string will be returned.

Returns

A string containing the selected portion of V1 text.

Example

```
getVariable("MBMTP", "Constant Value", "SMID", "Continue on Fail")
getVariable("MBMTV1", "Constant Value", "This is a test", "Continue on Fail")
getVariable("MBMTV2", "Constant Value", "6", "Continue on Fail")
getVariable("MBMTV3", "Constant Value", "4", "Continue on Fail")
// Run the BMT Script
executeShellCommand("MSP BuilderMultiTool", "Continue on Fail")
// this will return the sub-string "is a"
```

SLEN – Length of String

Returns the length of the string specified by V1.

Variables

MBMTV1 The string to evaluate.

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

An integer representing the length of the V1 text string.

Example

```
getVariable("MBMTP", "Constant Value", "SLEN", "Continue on Fail")
getVariable("MBMTV1", "Constant Value", "This is a test", "Continue on Fail")
getVariable("MBMTV2", "Constant Value", "[=BNULL=]", "Continue on Fail")
getVariable("MBMTV3", "Constant Value", "[=BNULL=]", "Continue on Fail")
// Run the BMT Script
executeShellCommand("MSP BuilderMultiTool", "Continue on Fail")
// The MBMTResult variable contains the command result - save it!
```

SINS – String Search - Position

Returns an integer representing the position in the V1 string where the V2 text is found. Returns 0 if the text being searched for is not found

Variables

MBMTV1 The string to evaluate.

MBMTV2 A string of text to search for in V1.

MBMTV3 OPTIONAL Integer – the offset position to begin the search.

Returns

An integer specifying the location where the text was found.

Example

Note the use of the escaped space character when defining MBMTV2 – this is important as leading/trailing spaces are normally stripped by the script, and we want to match the second "is", which is preceded by a space character. The initial "\s" will be replaced with an actual space character after the script collects and validates all of the supplied arguments.

```
getVariable("MBMTP", "Constant Value", "SINS", "Continue on Fail")
getVariable("MBMTV1", "Constant Value", "This is a test", "Continue on Fail")
getVariable("MBMTV2", "Constant Value", "\sis ", "Continue on Fail")
getVariable("MBMTV3", "Constant Value", "[=BNULL=]", "Continue on Fail")
// Run the BMT Script
executeShellCommand("MSP BuilderMultiTool", "Continue on Fail")
// this will return the value "5". If the leading space in V2 is removed, the result is "3".
```

SINB – String Search - Boolean

Returns TRUE (1) if the text specified in V2 is found in V1, otherwise returns FALSE (0). Similar to SINS except that it returns a Boolean instead of the position found.

MBMTV1 The string to evaluate.

MBMTV2 A string of text to search for in V1.

MBMTV3 OPTIONAL Integer – the offset position to begin the search.

Returns

An integer specifying the location where the text was found.

Example

See SINS

SREV – Reverse Text

Returns the string specified by V1 in reverse character order.

Variables

MBMTV1 The string to evaluate.

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

A string containing the V1 text in reverse order

Example

See SLEN

SSPL –Return Field From Delimited String

Splits a delimited string and returns a specific field. Fields are zero-based.

Variables

MBMTV1 The string to evaluate.

MBMTV2 The delimiter character or string. Usually one character but can be a string.

MBMTV3 The field to return – the first field number is 0 (zero).

Returns

The text string from the specified field of the delimited V1 string.

Example

```
getVariable("MBMTP", "Constant Value", "SSPL", "Continue on Fail")
getVariable("MBMTV1", "Constant Value", "Mach.site.x.customerID", "Continue on Fail")
getVariable("MBMTV2", "Constant Value", ".", "Continue on Fail")
getVariable("MBMTV3", "Constant Value", "3", "Continue on Fail")
// Run the BMT Script
executeShellCommand("MSP BuilderMultiTool", "Continue on Fail")
// The MBMTResult variable contains the command result - save it!
// This example returns the customer ID field from a machine group ID. The V1
// value would usually be defined by a system variable.
```

SSFC – Return Field Count of Delimited String

Returns the field count of the delimited string specified by V1. This can be used to determine the "last" field number or verify that the string has the correct number of fields.

Variables

MBMTV1 The string to evaluate.

MBMTV2 The delimiter character or string. Usually one character but can be a string.

MBMTV3 Ignored.

Returns

An integer representing the number of fields found in the delimited string. Returns 0 if no delimiters were found in the V1 string. Field counts match the 0-based field IDs used by SSPL, so a value of 0 means 1 field, 1 means 2 fields (0 & 1) were found, etc.

Example

```
getVariable("MBMTP", "Constant Value", "SSFC", "Continue on Fail")
getVariable("MBMTV1", "Constant Value", "Mach.site.x.customerID", "Continue on Fail")
getVariable("MBMTV2", "Constant Value", ".", "Continue on Fail")
getVariable("MBMTV3", "Constant Value", "[=BNULL=]", "Continue on Fail")
// Run the BMT Script
executeShellCommand("MSP BuilderMultiTool", "Continue on Fail")
// The MBMTResult variable contains the command result - save it!
// This example would return "3", indicating 4 fields numbered 0-3.
```

SRPL – Replace Text in String

Searches the string defined by V1 for the text specified by V2 and replaces it with V3.

Variables

MBMTV1 The string to evaluate.

MBMTV2 The text to be replaced.

MBMTV3 The replacement text.

Returns

The modified version of V1 where all occurrences of V2 are replaced with V3. No modifications are made if the text of V2 was not found.

Example

```
getVariable("MBMTP", "Constant Value", "SSPL", "Continue on Fail")
getVariable("MBMTV1", "Constant Value", "This is a text string for texting", "Continue on Fail")
getVariable("MBMTV2", "Constant Value", "text", "Continue on Fail")
getVariable("MBMTV3", "Constant Value", "test", "Continue on Fail")
executeShellCommand("MSP BuilderMultiTool", "Continue on Fail")
// The MBMTResult variable contains the command result - save it!
// Returns "This is a test string for testing"
```

SASC – Return ASCII Code of Character

SASC returns the ASCII code from the first character of the string defined by V1.

Variables

MBMTV1 The string to convert or the decimal value of the ASCII code. For SASC, only the first character of the string is significant.

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

SASC – an Integer value between 0 and 127.

Example

See SLEN.

SCHR – Convert Value to ASCII Character

SCHR returns the character specified by the ASCII code defined by V1,which should be in the range of 1-127. Values below 32 may not be usable.

Variables

MBMTV1 The string to convert or the decimal value of the ASCII code. For SASC, only the first character of the string is significant.

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

SCHR – a single ASCII character. Note that control codes (values below 32) may not be able to be processed by Kaseya.

Example

See SLEN.

SCLC & SCUC – Case Conversion

SCLC returns the string defined by V1 converted to lower case.
SCUC returns the string defined by V1 converted to upper case.

Variables

MBMTV1 The string to convert.

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

The modified version of V1 where all leading and trailing whitespace is removed.

Example

See SLEN.

Note that all string processing within the MSP Builder Multi-Tool is case-insensitive. These functions are provided for situations where case-sensitive processing is desired or required.

SCVS – Compare Two Version Strings

Compares two version strings defined by V1 and V2 and returns a tri-state integer value. Version strings typically contain multiple dot-delimited fields, such a "2.5.12.8247".

Variables

MBMTV1 The version string to evaluate. Must be delimited with periods.

MBMTV2 The version string to evaluate. Must be delimited with periods.

MBMTV3 Ignored.

Returns

A tri-state integer:

-1 V2 is less than V1
0 V2 is equal to V1
1 V2 is greater than V1

Example

```
getVariable("CVERSION", "File Content", "Version.txt", "Continue on Fail")
// The CVERSION value now contains "2.5.12.8247"
getVariable("MBMTP", "Constant Value", "SCVS", "Continue on Fail")
getVariable("MBMTV1", "Constant Value", "[=CVERSION=]", "Continue on Fail")
getVariable("MBMTV2", "Constant Value", "2.5.12.9025", "Continue on Fail")
getVariable("MBMTV3", "Constant Value", "[=BNULL=]", "Continue on Fail")
executeShellCommand("MSP BuilderMultiTool", "Continue on Fail")
// The MBMTResult variable contains the command result - save it!
// Returns "1" indicating that V2 is newer/higher than V1
If("#MBMTResult#", "equals", "1")
  // Upgrade is required… call the upgrade procedure!
```

Time and Date Functions

Many of the time and date functions make use of a "timestamp" format. Specifically, this is a DATE and TIME string that uses the format "YYYY/MM/DD hh:mm:ss". This timestamp format is fixed and represents an order of time values from greatest to smallest unit. For all functions requiring a timestamp as an input parameter, if only a date is provided, the time will default to midnight.

Another value used is a cTime value. This is normally the number of seconds from a defined time epoch, such as January 1, 1900. To work within any potential number size limits, these functions use a "moving epoch" of January 1 of the *current year*. Since most Service Desk time calculations are measured in hours if not minutes, this is not a significant limitation. Be aware, however, that references to days in the prior year will be represented with negative numbers.

TCTS – Current TimeStamp

Returns the current date and time in timestamp format (YYYY/MM/DD hh:mm:ss).

Variables

MBMTV1 Ignored.

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

A string containing the current date and time.

Example

```
getVariable("MBMTP", "Constant Value", "TCTS", "Continue on Fail")
getVariable("MBMTV1", "Constant Value", "[=BNULL=]", "Continue on Fail")
getVariable("MBMTV2", "Constant Value", "[=BNULL=]", "Continue on Fail")
getVariable("MBMTV3", "Constant Value", "[=BNULL=]", "Continue on Fail")
// Run the BMT Script
executeShellCommand("MSP BuilderMultiTool", "Continue on Fail")
// The MBMTResult variable contains the command result - save it!
```

TCDT – Current Date

Returns the current date as YYYY/MM/DD.

Variables

MBMTV1 Ignored.

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

A string containing the current date.

Example

See TCTS

TCTM – Current Time

Returns the current time as hh:mm:ss.

Variables

MBMTV1 Ignored.

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

A string containing the current time.

Example

See TCTS

TCMN – Current Month Number

Returns the current Month number – 1-12.

Variables

MBMTV1 Optional – if True, the value is returned as a two-digit value with leading zero where appropriate.

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

A value representing the number of the current month.

Example

See TCTS

TCMO – Current Month Name

Returns the current Month Name.

Variables

MBMTV1 Optional – if True, the name is returned as a 3-char ALL CAP abbreviation.

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

A string containing the full name of the current month, or a 3-letter CAPs abbreviation (ie: JAN).

Example

See TCTS

TCDN – Current Day of Month

Returns the day of the current month (1-31).

Variables

MBMTV1 Optional – if True, the value is returned as a two-digit value with leading zeroes.

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

An integer value between 1 and 31 representing the current day of the current month.

Example

```
getVariable("MBMTP", "Constant Value", "TCDN", "Continue on Fail")
getVariable("MBMTV1", "Constant Value", "Yes", "Continue on Fail")
getVariable("MBMTV2", "Constant Value", "[=BNULL=]", "Continue on Fail")
getVariable("MBMTV3", "Constant Value", "[=BNULL=]", "Continue on Fail")
// Run the BMT Script
executeShellCommand("MSP BuilderMultiTool", "Continue on Fail")
// The MBMTResult variable contains the command result - save it!
// This will return "03" if MBMTV1 is True, otherwise will return "3"
```

TCYN – Current Year

Returns the current year as a 4-digit number.

Variables

MBMTV1 Ignored.

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

A 4-digit integer representing the current year.

Example

See TCTS

TJDT – Current Julian Day

Returns the current Julian day – the current day of the year – 1-365.

Variables

MBMTV1 Ignored.

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

An integer value representing the Julian day.

Example

See TCTS

TCWD - Current Weekday Name

Returns the current weekday name abbreviation. The string is returned as all capital letters.

Variables

MBMTV1 Ignored.

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

An all-caps string containing the current day name abbreviation, SUN, MON, TUE, WED, THU, FRI, or SAT.

Example

See TCTS

TDY0 – Current Weekday Number (Sunday=0)

Returns the current weekday number with Sunday represented as 0 (zero). Monday is 1, Saturday is 6. This is the common computer representation of weekday numbers.

Variables

MBMTV1 Ignored.

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

An integer in the range of 0-6.

Example

See TCTS

TDY7 – Current Weekday Number (Sunday=7)

Returns the current weekday number with Sunday represented as 7. Monday is 1, Saturday is 6. This format keeps the weekend days adjacent for easier calculation of weekday/weekend events.

Variables

MBMTV1 Ignored.

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

An integer in the range of 1-7.

Example

See TCTS

TIWE – Is WeekEnd

Returns TRUE (1) if the current day/time is a Saturday or Sunday (weekend), otherwise returns a value of FALSE (0).

Variables

MBMTV1 Ignored.

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

An integer in the range of 0-1.

Example

See TCTS

TGDP – Get Date Part

Returns the Date part of a timestamp specified by V1.

Variables

MBMTV1 Timestamp string.

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

A string containing the date part of the supplied timestamp.

Example

```
getVariable("MBMTP", "Constant Value", "TGDP", "Continue on Fail")
getVariable("MBMTV1", "Constant Value", "2015/11/23 09:53:28", "Continue on Fail")
getVariable("MBMTV2", "Constant Value", "[=BNULL=]", "Continue on Fail")
getVariable("MBMTV3", "Constant Value", "[=BNULL=]", "Continue on Fail")
// Run the BMT Script
executeShellCommand("MSP BuilderMultiTool", "Continue on Fail")
// The MBMTResult variable contains the command result - save it!
// This will return "2015/11/23"
```

TGTP – Get Time Part

Returns the Time part of a timestamp specified by V1.

Variables

MBMTV1 Timestamp string.

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

A string containing the time part of the supplied timestamp.

Example

See TGDP

TCVT – cTime Conversion

Converts a timestamp specified by V1 into a modified cTime numeric value.
Converts a modified cTime value specified by V1 into a timestamp string.

This function allows simple conversion between timestamp and cTime type values. This is the basis for all time calculations. For example, to document the time 15 minutes from now, you would use the TCVT without arguments to obtain the current cTime value. This result would then be placed into the V1 value, 900 placed into the V2 value, and the ADD function invoked. This would return the cTime with 900 seconds (15 minutes) added to it. Placing this returned value into V1 and calling TCVT again would return a timestamp 15 minutes from the current time.

This is an excellent example of how all of the MSP Builder Multi-Tool functions can work together.

Variables

MBMTV1 — Optional – Date, Timestamp string or modified cTime numeric value. If a date without a time part is provided, it defaults to midnight of the current day. If the value is empty, it defaults to the current Date and Time. Specifying a value of 0 returns the timestamp of the current / defined epoch (default is January 1 of the prior year).

MBMTV2 — Optional – a Date value representing an alternate Epoch value. The default Epoch value is 1/1 of the prior year and was chosen to keep the values reasonably small. This rarely needs to be specified unless you are working with an existing cTime value that uses a standard Epoch value. (Unix Epoch: 1970/01/01, Windows Epoch: 1900/01/01).

MBMTV3 — Ignored.

Returns

A number representing the supplied timestamp string as a cTime value.
A string containing the timestamp converted from the supplied cTime value.

Example

Obtain a timestamp that is 15 minutes (900 seconds) in the future.

```
// Get the current cTime value – with no args, the current cTime is returned
getVariable("MBMTP", "Constant Value", "TCVT", "Continue on Fail")
getVariable("MBMTV1", "Constant Value", "[=BNULL=]", "Continue on Fail")
getVariable("MBMTV2", "Constant Value", "[=BNULL=]", "Continue on Fail")
getVariable("MBMTV3", "Constant Value", "[=BNULL=]", "Continue on Fail")
// Run the BMT Script
executeShellCommand("MSP BuilderMultiTool", "Continue on Fail")
// The MBMTResult variable contains the command result – save it!
// .
// Add 900 seconds to the cTime value
getVariable("MBMTP", "Constant Value", "ADD", "Continue on Fail")
getVariable("MBMTV1", "Constant Value", "[=MBMTResult=]", "Continue on Fail")
getVariable("MBMTV2", "Constant Value", "900", "Continue on Fail")
// Run the BMT Script
executeShellCommand("MSP BuilderMultiTool", "Continue on Fail")
// The MBMTResult variable contains the command result – save it!
//.
// convert the returned cTime value to a timestamp
getVariable("MBMTP", "Constant Value", "TCVT", "Continue on Fail")
getVariable("MBMTV1", "Constant Value", "[=MBMTResult=]", "Continue on Fail")
getVariable("MBMTV2", "Constant Value", "[=BNULL=]", "Continue on Fail")
// Run the BMT Script
executeShellCommand("MSP BuilderMultiTool", "Continue on Fail")
// The MBMTResult value contains the timestamp for 15 minutes in the future!
```

Using a negative number or the SUB function will return a timestamp from the past instead of the future.

TDIF – Time Difference

Computes the difference between two timestamps. Returns the time difference in seconds (default), minutes, hours, days, or years. Selecting a different format will return a double-precision number. This assumes that the start time is before the end time, but if the end time is instead before the start time (calculating an end in the past), a negative result will be returned.

This is a powerful function that allows calculating future target times for scheduling events, or determining the time that has elapsed since an event occurred.

Variables

MBMTV1 — Timestamp string representing the start time. The special term "today" can be used to specify midnight of the current day, or "now" for the current timestamp. If a date is provided without a time component, the time is set to midnight (00:00:00).

MBMTV2 — Timestamp string representing the end time. Defaults to the current time if not specified. The special term "today" can be used to specify midnight of the current day. If a date is provided without a time component, the time is set to midnight (00:00:00).

MBMTV3 — OPTIONAL – String – one of:
M – Return time difference in minutes.
H – Return time difference in hours.
D – Return time difference in days.
Y – Return time difference in years.
If not specified, the elapsed time is returned in seconds.

Returns

A number representing the interval between two timestamps, using the defined or default unit or measure. Units of measure other than seconds will return a value with a decimal (fractional) component.

Example

One example might take the time that the ticket was opened and capture the elapsed time at different points in the process to develop a grading system, or to see where process improvements might be possible. Another might be used to adjust goal times for procedures based on the amount of time already taken to process a ticket.

TNTO – Next Time Occurrence

Computes the timestamp of the next occurrence of a time, such as "Next Tuesday at 2pm" or "Third Thursday at 1am". This function simplifies planning of regularly occurring events, or targeting the next available time for maintenance, reboots, etc. The longest interval supported is roughly 1 month.

Variables

MBMTV1 — The week in the month - a numeric value from 0 to 5, where 0 represents "next available", 1-4 represent "first" through "fourth", and 5 represents "Last".

MBMTV2 — The day of the week – a numeric value from 0 to 7, where 0 represents "next available" and 1-7 represent Monday through Sunday. (1=Monday, 7=Sunday)

MBMTV3 — The time to run, as HH:MM:SS.

Returns

A timestamp representing the next available date and time of the desired event.

Example

Return the timestamp when 14:30 will next occur – MBMTV1=0, MBMTV2=0, MBMTV3=14:30:00. Returns a timestamp with the current date if the current time is prior to 14:30, or tomorrow's date if it is later than 14:30.

Return the timestamp of the next occurrence of Tuesday at 3pm – MBMTV1=0, MBMTV2=2, MBMTV3=15:00:00.

Return the timestamp of the next occurrence of the Last Sunday of the month at 2am – MBMTV1=5, MBMTV2=7, MBMTV3=02:00:00.

NOTE: When creating a Next Time Occurrence value, be aware that if the target time has not yet arrived and the current day meets the specified criteria, the current date will be returned. For example, if you specify Next Friday at 14:00 on a Friday at 10:00, the targeted time will be 4 hours away.

TITR & TIDR – In Time/Date Range

Two related functions that work with time only (TITR) or Time-Date timestamps (TIDR). They determine if a specific (or current) time is within a time range specified by two time or timestamp values. This can be used to determine coverage schedules (such as - if a customer does not have extended coverage and the event time falls within extended coverage hours, then do not page the on-call team) or decide if specific goals are being met.

TITR determines if the time is between a specific pair of time values (no date). Any date component will be stripped for this comparison. This function allows simple timespan checks of a sliding 24-hour period regardless of the date.

Note that if the END time is less than the START time, it will be assumed that the end time occurs during the following day. This allows direct determination of the current time, for example, being between 5pm (17:00) of the current day and 6am (06:00) of the following day. The maximum time resolution of TITR is 24 hours. Use TIDR if longer time spans are required.

TIDR determines if the check time is between a specific pair of timestamps. This can be used to span long date ranges. The start timestamp *must* occur before the end timestamp or the function will return false (0). Note that TIDR checks the entire time span between the timestamps. If you specify "2015/01/01 08:00:00" and "2015/01/31 17:00:00", a check timestamp of "2015/01/17 23:40:02" would return True, indicating that it occurred between the two specified timestamps.

Variables

MBMTV1 — Time or timestamp string representing the starting point of the comparison.
TITR – Any date component is removed. Will report an error if no time data is found.
TIDR – Defaults to midnight if no time is provided. Will report an error if no date data is found.

MBMTV2 — Time or timestamp string representing the end point of the comparison
TITR – Any date component is removed. Will report an error if no time data is found.
TIDR – Defaults to midnight if no time is provided. Will report an error if no date data is found.

MBMTV3 — Time or timestamp to compare against the start and end values.
TITR – Any date component is removed. Defaults to the current time if not specified. If the single digit "1" is supplied, the time will default to the current time, and the result will be true only if the current day is a weekday. This is especially useful for determining helpdesk coverage hours running Monday through Friday.
TIDR – Defaults to midnight of the current date if no data is provided. Defaults to midnight if only a date (no time) is provided.

Returns

An integer value in the range 0-1. A value of 1 indicates that the V3 time is within the time or timestamp range defined by V1 and V2. If invalid date or time values are provided, an error message will be returned as "ERROR <message>. The message will identify the error, such as "No Time#" or "No Date#". The "#" indicates which value was invalid.

Example

The most common use of **TITR** is to determine if the help desk is active or not. If the test succeeds, it would mean that the help desk is not manned, and that high priority tickets will require the on-call staff to be notified. A second level test might be to check for extended coverage hours, so that on-call notifications are made only within certain hours based on the customer's coverage hours. TIDR would be useful for calculating longer durations spanning multiple days, such as "was the ticket completed within the target time".

The examples below show two methods of determining if the current time is outside of 8:00-17:00 or within that time range during the week.

```
// Is the current time outside of helpdesk coverage hours?
// this method does not take weekend/weekday into account. The
// TIWE can be tested to be 0 prior to this logic to further
// restrict the evaluation.
getVariable("MBMTP", "Constant Value", "TITR", "Continue on Fail")
// help desk coverage ends
getVariable("MBMTV1", "Constant Value", "17:00:00", "Continue on Fail")
// help desk coverage starts
getVariable("MBMTV2", "Constant Value", "08:00:00", "Continue on Fail")
// blank to default to current time
getVariable("MBMTV3", "Constant Value", "[=BNULL=]", "Continue on Fail")
// Run the BMT Script
executeShellCommand("MSP BuilderMultiTool", "Continue on Fail")
// The MBMTResult variable contains the command result - save it!
If checkVariable(MBMTResult", "equals", "1")
  // current time is when the help desk is closed, enable Paging flag
  getVariable("DoPage", "Constant Value", "1", "Continue on Fail")

// Is the current time inside of helpdesk coverage hours (8am-5pm M-F)?
 getVariable("MBMTP", "Constant Value", "TITR", "Continue on Fail")
// help desk coverage ends
getVariable("MBMTV1", "Constant Value", "08:00:00", "Continue on Fail")
// help desk coverage starts
getVariable("MBMTV2", "Constant Value", "17:00:00", "Continue on Fail")
// blank to default to current time
getVariable("MBMTV3", "Constant Value", "1", "Continue on Fail")
// Run the BMT Script
executeShellCommand("MSP BuilderMultiTool", "Continue on Fail")
// The MBMTResult variable contains the command result - save it!
If checkVariable(MBMTResult", "equals", "1")
  // current time is when the help desk is active, disable Paging flag
  getVariable("DoPage", "Constant Value", "0", "Continue on Fail")
```

TDLY – Time Delay

Causes the program to sleep for the defined number of seconds. For safety, the delay is limited to 900 seconds (15 minutes).

Variables

MBMTV1 An integer value specifying the number of seconds to delay, in the range of 1-900. Input values larger than 900 are forcibly set to 900.

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

Always returns 1 (True) at the completion of the delay.

Logic Functions

A collection of logic functions that permit Boolean tests, logic and binary comparisons, and inversion.

The following values will evaluate as True in all of the logic comparisons:

- Any non-zero numeric value, positive or negative.
- "T" or "True".
- "Y" or "Yes".
- "On".

LRTF – Return True/False

Return a logical True (1) or False (0) based on a set of text or numeric values.

Variables

MBMTV1 The value to test.

MBMTV2 Optional – comma-delimited list of additional text values that will evaluate to true. For example, if V1 contains "Ja" and V2 contains "Ja,Si", the result would evaluate as "true". *Use caution with this parameter as it is possible to override values that normally evaluate to False – if V2 contains "No", then either "Yes" or "No" would return "True"!*

MBMTV3 Ignored.

Returns

A number in the range of 0-1.

Example

A value can be read from a file or entered by a user. Rather than using multiple IF statements, the LRTF function can compare a value against a series of values that represent a True state. After calling this function, the Kaseya procedure logic simply needs to check for 0 (False) or 1 (True).

LNOT – Logical NOT (Inverse)

Return a logical inverse of a value, returning 1 for a 0 value, and 0 for any non-zero input value. Text values are treated as numerically zero, except those values recognized as Boolean.

Variables

MBMTV1 The numeric value to evaluate. Text is treated as 0, which will return 1.

MBMTV2 Ignored.

MBMTV3 Ignored.

Returns

A number in the range of 0-1.

Example

```
getVariable("MBMTP", "Constant Value", "LNOT", "Continue on Fail")
getVariable("MBMTV1", "Constant Value", "False", "Continue on Fail")
getVariable("MBMTV2", "Constant Value", "[=BNULL=]", "Continue on Fail")
getVariable("MBMTV3", "Constant Value", "[=BNULL=]", "Continue on Fail")
// Run the BMT Script
executeShellCommand("MSP BuilderMultiTool", "Continue on Fail")
// The MBMTResult variable contains the command result - save it!
// The result will return "1", the inverse of the value supplied.
```

LAND – Logical AND

Return a logical result of an AND of values MBMTV1, MBMTV2, and MBMTV3. Text values are treated as numerically zero, except those values recognized as Boolean.

Variables

MBMTV1 The numeric or Boolean value to evaluate.

MBMTV2 The numeric or Boolean value to evaluate.

MBMTV3 Optional - The numeric or Boolean value to evaluate.

Returns

A number in the range of 0-1.

Example

```
getVariable("MBMTP", "Constant Value", "LAND", "Continue on Fail")
getVariable("MBMTV1", "Constant Value", "True", "Continue on Fail")
getVariable("MBMTV2", "Constant Value", "No", "Continue on Fail")
getVariable("MBMTV3", "Constant Value", "0", "Continue on Fail")
// Run the BMT Script
executeShellCommand("MSP BuilderMultiTool", "Continue on Fail")
// The MBMTResult variable contains the command result - save it!
// The result will return "0" because at least 1 value is "false".
```

LOR – Logical OR

Return a logical result of an OR of values MBMTV1, MBMTV2, and MBMTV3. Text values are treated as numerically zero, except those values recognized as Boolean.

Variables

MBMTV1 The numeric or Boolean value to evaluate.

MBMTV2 The numeric or Boolean value to evaluate.

MBMTV3 Optional - The numeric or Boolean value to evaluate.

Returns

A number in the range of 0-1.

Example

```
getVariable("MBMTP", "Constant Value", "LOR", "Continue on Fail")
getVariable("MBMTV1", "Constant Value", "True", "Continue on Fail")
getVariable("MBMTV2", "Constant Value", "No", "Continue on Fail")
getVariable("MBMTV3", "Constant Value", "0", "Continue on Fail")
// Run the BMT Script
executeShellCommand("MSP BuilderMultiTool", "Continue on Fail")
// The MBMTResult variable contains the command result - save it!
// The result will return "1" because at least 1 value is "true".
```

Network Functions

A Collection of functions related to TCP/IP networking, capable of performing NSLookup queries and determining if an IP address is in a specific network.

NNSL – NSLookup Name or IP

Performs an NSLookup query (from the Kaseya server) for a specified hostname or IP address. Note that only public hostnames/IPs can be resolved unless your Kaseya server is used only for internal (single customer) monitoring. Further, be aware that this function is limited by the network security setting in effect in the environment where the Kaseya server resides.

If multiple IPs are returned, they will be delimited with a semicolon (;).

Variables

MBMTV1 — The hostname or IP address to resolve.

MBMTV2 — Optional – numeric digit. This value must be specified if MBMTV3 is defined.
0 – Return both hostname and IP address, delimited with a comma.
1 – Return just the hostname part of the resolved value.
2 – Return just the IP Address part of the resolved value.

MBMTV3 — Optional – numeric digit.
4 – Return only IPV4 addresses.
6 – Return only IPV6 addresses.
Default (not specified) is to return all addresses/types.

Returns

A string with the hostname,IP, hostname, or IP address. Will return “ERROR Lookup Failed” if it was unable to resolve the requested name or IP.

Example

```
getVariable(“MBMTP”, “Constant Value”, “NNSL”, “Continue on Fail”)
getVariable(“MBMTV1”, “Constant Value”, “www.mysite.com”, “Continue on Fail”)
getVariable(“MBMTV2”, “Constant Value”, “2”, “Continue on Fail”)
getVariable(“MBMTV3”, “Constant Value”, “[=BNULL=]”, “Continue on Fail”)
// Run the BMT Script
executeShellCommand(“MSP BuilderMultiTool”, “Continue on Fail”)
// The MBMTResult variable contains the command result – save it!
// The result will contain the IP Address of the resolved hostname.
```

NISN – IP In Network Subnet

Returns True if the specified IPV4 address is within the network/mask (CIDR format) specified by V2.

Variables

MBMTV1 The IP address to verify.

MBMTV2 The network and mask in CIDR format (A.B.C.D/##).

MBMTV3 Ignored.

Returns

True (1) if the IP resides within the specified subnet, False (0) if it does not.

Example

```
   getVariable("MBMTP", "Constant Value", "NISN", "Continue on Fail")
   getVariable("MBMTV1", "Constant Value", "172.16.12.18", "Continue on Fail")
   getVariable("MBMTV2", "Constant Value", "172.16.8.0/22", "Continue on Fail")
   getVariable("MBMTV3", "Constant Value", "[=BNULL=]", "Continue on Fail")
   // Run the BMT Script
   executeShellCommand("MSP BuilderMultiTool", "Continue on Fail")
// The MBMTResult variable contains the command result - save it!
// The result will contain "0" (False).
```

Logging and Debugging

By default, every invocation of the MSP Builder Multi-Tool writes a single line to a log file (MBMTU.LOG). This is useful when troubleshooting procedures. Each log entry contains the date and time, the calling source (optional), the name of the function, the three data values, and the result. This can help verify that the proper values are being sent to and returned from the BMT program.

The location of the log file defaults to the "Logs" subfolder in the script directory but this can be overridden with a system environment variable "MBMTULOGPATH". This should point to a folder where the log can be written to. The log file cannot be specified, and the value must be terminated with a backslash.

Logging can be minimized to recording just errors by replacing the MBMTU.KIX script with the MBMTUNL.KIX script. For convenience, you may want to install both copies to your Kaseya server, naming one MBMTU_FULL.KIX and the other MBMTU_NL.KIX, and then copy one or the other to "MBMTU.KIX". The "No Logging" version will suppress the detail result messages *when they are successful*, but logs will still be written any time an error in processing occurs. If full logging is used, the file should be trimmed regularly.

Source ID

Another debugging feature is the ability to define a "Source ID", which can identify the procedure that generated the log entry. This can be implemented simply by placing a prefix in front of the function and using a colon (":") between the identifier prefix and the function. The prefix should be limited to letters, numbers, and an underscore and should readily identify the source of the function call. For example, you could simply prefix functions called from your Ticket Intake procedure with "TIXIN:", or you could identify the source and the line or code block, such as "TIXN_14:" for the call from line 14 of the ticket intake procedure.

Special Logging

Debugging messages can be helpful while developing Service Desk procedures. Until now, developers were limited to sending emails with notes and values, or writing messages to the ticket via "Add Note". This becomes difficult when developing intake procedures that may cancel the ticket, as these notes are lost if the ticket is cancelled. The MSP Builder Multi-Tool provides a unique debugging feature that allows up to 3 lines of text to be written to the MBMTU.LOG file on the Kaseya server. The WLOG function will write a delimiter line, up to 3 message lines (one each for MBMTV1, MBMTV2, and MBMTV3), and a second delimiter line. The values may contain any message or variables that you wish to record.

Log Management

No automatic log rotation is done by the MSP Builder Multi-Tool, but log management functionality is built-in via command-line arguments. The MBMTU script can be invoked via the task scheduler using a simple batch file, similar to:

```
@Echo Off
Set CmdPath=D:\Kaseya\MyTools
%CmdPath%\kix32.exe %CmdPath$\MBMTU.KIX --LOG:<code>
```

"<code>" must be one of:

"1" The log is renamed from *MBMTU.log* to *MBMTU_#-DAY.log*. Any prior file with this name is removed. At most, 1 week of logs will be retained, each having a DAY identifier as part of the name. The "#" represents the day number (Monday = 1, Sunday=7). This is followed by the day abbreviation. The number prefix insures that the file names are displayed in sequence.

"2" The log is renamed from *MBMTU.log* to *MBMTU_<YYYYMMDD.log*. Any prior file with this name is removed. This creates one log per day of operation, and the accumulated daily logs should be managed by external means. If limited logging is used, then this method would be acceptable if scheduled to run weekly.

"3" The log is renamed from *MBMTU.log* to *MBMTU_OLD.log*. Any prior file with this name is removed. This mode requires the least space, with only one prior day's events preserved.

If the code is not specified or is invalid, no actions will be taken. Setting up some form of log management is highly recommended to conserve disk space and minimize any performance impact due to writing large log files.

Note: The account used to run the scheduled task should have Modify rights granted on the Logs folder. This account does not need administrator access, and for optimal security, should be a regular user type account. No additional access to any other folder is needed by this account.

Customization

Do you have a specific need not met by the MSP Builder Multi-Tool? Contact us with your requirements at support@mspbuilder.com. Most custom features can be added quickly and for a minimal one-time fee. Customization fees may be waived for any "why didn't we think of that?" functions that would have wide general appeal.

The MSP Builder Multi-Tool is a "stateless" application. It cannot track values from one invocation to another. As our examples show, the returned value must be saved and then moved back into one of the Value variables for further processing. Since the same program may be invoked by many concurrent procedures, we cannot provide stateful processing capabilities.

Evaluation & Debug Mode

The MSP Builder Multi-Tool supports an evaluation mode. A subset of the functions are available from within the Kaseya Service Desk. This will allow testing of the integration with your Kaseya platform.

Running the tool from a command prompt on a workstation causes the tool to operate in DEBUG mode, which displays the input, argument count, and results, but will not provide useful output to Kaseya. This allows the developer to become familiar with of all MMBMTU functions.

Support Requests

MSP Builder Technologies is an MSP in the NYC metro area, so we understand the Service Desk automation requirements of MSPs. If you are stuck trying to make something work, drop us a line at MBMT@mspbuilder.com. We'll get back to you via email with a solution within 3-5 days. We can also provide custom (fee based) procedure coding for those unusual or complex situations.

Other Products & Services

The MSP Builder Multi-Tool is just one part of the MSP Builder MSP solution.

- The RMM Suite complete turn-key Service Desk solution with monitors, daily maintenance tools, automatic remediation capability, KNM integration, and even a dial-out Incident Notification System (BINS) for round-the-clock alerting of priority events. The BINS system honors customer coverage classes (standard, extended, and full) and provides immediate email notification of all priority 1-2 events.
- Our Log Management utility provides a standard utility for managing any type of application log file, including Event Logs and web logs. Built-in archiving (with Zip-file compression) allows local storage for specified periods. Runs as a system service or a standalone app (manually or via system scheduler) for the utmost in flexibility.
- The Universal Login Script provides a fast and reliable, *code free* method for configuring the user environment by mapping drives, installing network printers, displaying messages, and running customization scripts. With more than a dozen methods for resource authorization, the processing of resources provides nearly unlimited flexibility. A GUI is available to manage the configuration file for a consistent management.
- Our "What are Managed Services?" video explains what an MSP offers and why it benefits a company to employ them! Short yet informative, it can be embedded in your web site as a marketing tool or used in training for customer onboarding.

Visit www.mspbuilder.com for the full story and latest information on our full product line!

This page intentionally blank.

Implementation Notes

This section will provide some notes and ideas for implementing Kaseya Service Desk tasks using the MSP Builder Multi-Tool. Examples will be simplified to logic flows and pseudo-code, as the exact commands have been documented in the Function Description section.

Math Functions

Most math functions will operate against a specific value that is being tracked in a variable or custom field. This is the typical logic flow, using the INC function to increment a value. Note that the CLT function is used to check the value to avoid Kaseya's string-based comparisons. Variable assignments are shown in a simplified form for brevity but would use getVariable() functions in the procedure.

```
// set the count value – this can be hard coded or be loaded from a custom value ([$Count$])
COUNT = 8
// increment a count value
MBMTP = INC
MBMTV1 = [=COUNT=]
MBMTV2 = "[=BNULL=]"
MBMTV3 =  "[=BNULL=]"
execShellCommand(MSP BuilderMultiTool)
// save the result into Count, then put COUNT back into V1
COUNT = "[=MBMTResult=]"
MBMTV1 = "[=COUNT=]"
// do some other processing here…
// prepare to perform a comparison, the current count is already in V1
MBMTP = "CLT"
MBMTV2 = "10"
execShellCommand(MSP BuilderMultiTool)
// the result will be TRUE (1) if the comparison succeeded (COUNT < 10)
IF "[=MBMTResult=]"  = 1
   // maybe do more stuff…
// Store the count into the custom field, if necessary
```

This should illustrate how the MBMTV# variables are used with multiple function calls.

Comparisons

All of the comparison functions return a 0 (false, compare failed) or 1 (true, compare succeeded). This allows a basic If statement in Kaseya to specifically match a 0 or 1, which will work without the issues related to text-based comparisons. If the comparison "4 < 10" is done via string, as Kaseya normally does, the values are compared on a character by character basis, and 4 is greater than 1, so the compare fails when it should actually succeed if it had been compared mathematically. The MSP Builder Multi-Tool forces both values to real numbers before performing the comparison. The True/False that is returned can be reliably checked using Kaseya's string compare ("1" will always match "1" and will never match "0").

The equality (CEQ) and inequality (CNE) functions operate with string or numeric values. When both values are numeric, they are converted to double-precision decimal values and then compared. If either of the supplied values is not numeric, a text-based comparison is performed. This selection of text or numeric comparison is automatic.

String Manipulation

Most of the string manipulation functions provided by the MSP Builder Multi-Tool are straight-forward to anyone with basic programming or scripting knowledge. There are two functions that deal with delimited strings that bear some additional discussion.

A delimited string is any text that can be broken into distinct parts using a specific character. The most obvious of these would be the machine-group ID. For discussion, let's say that you organize your agents by customer, location, and type (server or workstation). Thus, "PC-1234" might have a machine-group ID of pc-1234.workstation.dallas.univtexas. This string is delimited with periods, making it easy to break into sections. If we wanted to identify the location, we would need the third field (dallas). Since computers start numbering from 0, this would actually be field number 2.

```
MBMTP = SSPL
MBMTV1 = [$machine.groupname$]
MBMTV2 = "."
MBMTV3 = 2
execShellCommand(MSP BuilderMultiTool)
// The result variable MBMTResult will contain "dallas"
```

This is pretty straight-forward, but what happens when you sometimes have an additional sub-group? Some of your agents might have 4 fields while others might have 5. To obtain the location info, you need the *next to last* field data. In this example, the agent name is "pc-98.workstation.biotech.dallas.univtexas".

The SSFC function will return the zero-based count of fields in a delimited string. Thus, to find the location held in the next to last field, you would perform the following steps. Note that it takes 3 calls to the MSP Builder Multi-Tool. One to obtain the field count, one to decrement it, and one to extract the data.

```
MBMTP = SSFC
MBMTV1 = [$machine.groupname$]
MBMTV2 =  "."
MBMTV3 = " "
execShellCommand(MSP BuilderMultiTool)
// The result var MBMTResult will contain "4" (5 fields, 4 being the last)
// Save the result and then decrement it
MBMTP = DEC
MBMTV1 = [=MBMTResult=]
execShellCommand(MSP BuilderMultiTool)
// the result will now contain 3 for this situation, the next to last field #
// now extract the field data
MBMTP = SSPL
MBMTV1 = [$machine.groupname$]
MBMTV2 = "."
MBMTV3 = [=MBMTResult=]
execShellCommand(MSP BuilderMultiTool)
// The result var MBMTResult will contain "dallas"
```

Note that if you needed a different field, such as the second-to-last, you would use SUB to subtract two rather than DEC to simply decrement the field count.

Note that these examples use a single-character delimiter, but multi-character delimiters are permitted. Consider breaking the following string into two parts to obtain the error code: "Error while performing input operation: 2055". In this case, using "operation:\^s" as the delimiter and requesting field 1 would return the value "2055". Note that the trailing space uses the escape character to specify the trailing space.

More String Manipulation

Sometimes you will need to decide on a process to perform based on whether specific text exists in a string. There are two functions available to help with this. The SINS function can tell you not only *if* a text value is present in a text string, but *where*. The SINB function just returns True (1) if the text was found. If the text you are searching for is not found, both functions return 0 (zero).

Both of these functions accept an optional third parameter that indicates where (by character count) the search should start. The default is to search from the beginning of the string. Note that the SINS function returns the position that the text was found starting from the beginning of the string, not where it started searching from!

MBMTP = SINB
MBMTV1 = [$body$]
MBMTV2 = "Fault"
MBMTV3 = "36"
execShellCommand(MSP BuilderMultiTool)
// the result will contain "1" if "fault" was found in the ticket body text *after* the 36th character.

Let's assume that the body contains a message "Fault: disk0 in array DA2 has failed", and you want to identify the disk that failed.

- Use the SINS function to search for "Fault:\s " in the ticket body. This will return the character position where the "F" was found. Note that the search string contains the colon and space to better identify the desired position, and that the space is *escaped* with "\s". This minimizes falsely finding a message containing "fault". When searching, be as specific as possible!
- Use the ADD function to add 7 to the number that was returned. This will skip over the "Fault: " part of the text (and everything before it). The returned value will be pointing to the character position where "disk0…" begins.
- Use the SMID function to select all the remaining text starting with the position of "disk0…"
  SMID [$body$] <position_to_start> <length>
  < position_to_start> was calculated by the result of the ADD function. <length> is optional – leave it blank to return all of the remaining text, or specify a value to control the number of characters you are interested in.

If the SMID function is restricted to, say, 50 characters, the result will identify both the disk number and array ID (along with some additional and possibly unneeded text). This is a *delimited* string (the spaces are delimiters) and the SSPL function can now be used to extract the disk number (field 0) and array ID (field 3).

Time Calculations

The time functions provide some of the more sophisticated capabilities that the Multi-Tool has to offer. A common process is sending an escalation alert to on-call staff when the help desk is closed. This example will allow a call escalation if the customer is in coverage hours and the help-desk is not operating.

```
getVariable(MBMTP, Constant Value, TIWE)
getVariable(MBMV1, Constant Value, [=MBNULL=])
getVariable(MBMV2, Constant Value, [=MBNULL=])
getVariable(MBMV3, Constant Value, [=MBNULL=])
executeShellCommand(MSP_BuilderMultiTool)
// save the result in an "Is Weekend" flag variable
getVariable(Is_WEnd, Constant Value, [=MBMTResult=])

// Set a flag to determine if the customer is within their coverage time
If isWithinCoverage()
  getVariable(NotifyNow, Constant Value, 1)
Else
  getVariable(NotifyNow, Constant Value, 0)

// Get the helpdesk start and end values for weekday or weekend/holiday
// these are global SD variables
If checkVariable(#Is_WEnd#, equals, 0)
  getVariable(HD_Start, Constant Value, [=RMM_HD_WDayS=])
  getVariable(HD_End, Constant Value, [=RMM_HD_WDayE=])

Else
  getVariable(HD_Start, Constant Value, [=RMM_HD_WEndS=])
  getVariable(HD_End, Constant Value, [=RMM_HD_WEndE=])

// call a SQL query to check if today is a defined holiday
executeSqlQuery(GetHolFlag, IsHoliday)

// disable notify if helpdesk is in operating hours and not a holiday
If checkVariable(GetHolFlag, equals, 0)
  // is the current time within helpdesk operating hours?
  getVariable(MBMTP, Constant Value, TITR)
  getVariable(MBMV1, Constant Value, [=HD_Start=])
  getVariable(MBMV2, Constant Value, [=HD_End=])
  getVariable(MBMV3, Constant Value, [=MBNULL=])
  executeShellCommand(MSP_BuilderMultiTool)
  // the result will be 1 if in operating hours
  getVariable(HDActive, Constant Value, [=MBMTResult=])
  // if operating, disable NotifyNow
  If checkVariable(HDActive, equals, 1)
    getVariable(NotifyNow, Constant Value, 0)
```

At this point, the NotifyNow variable will be true if the alert occurred while the customer is within their coverage time *and* the help desk is not staffed. An escalation process can be performed if NotifyNow is true.