Blockland Docs Documentation

Release 19

Blockland community

November 29, 2016

Contents

1 Strings **3**

1.1 Manipulation 3

1.2 Processing 4

1.3 Words 4

1.4 Tokenizing 5

2 Math **7**

3 Vectors **9**

3.1 Examples 9

4 Return to Blockland **11**

4.1 Examples 12

5 Indices and tables **15**

Topics:

CHAPTER 1

Strings

1.1 Manipulation

getSubStr (*%string*, *%index*[, *%length*])
Returns the characters in %string from %index, *optionally* only returning %length characters from the start of %index.

strStr (*%haystack*, *%needle*)
Searches for %needle in the %haystack *with* case-sensitivity. Returns the index of the beginning of %needle in %haystack on success and -1 when it cannot be found.

strReplace (*%str*, *%old*, *%new*)
Replaces all instances of %old in %str with %new and returns the result.

```
==> strReplace("Hello, World!","World","Jeff");
Hello, Jeff!
```

strLwr (*%str*)
Returns all characters in %str in lower-case form.

strUpr (*%str*)
Returns all characters in %str in upper-case form.

stripChars (*%str*, *%chars*)
Removes every instance of any character in %chars from %str and returns the result.

stripMLControlChars (*%str*)
Removes any Torque Markup Language tags from %str and returns the result.

strTrim (*%str*)
Removes any white space from the left and right of %str and returns the result.

trim (*%str*)
Removes any white space from the left and right of %str and returns the result.

ltrim (*%str*)
Removes any white space from the left of %str and returns the result.

rtrim (*%str*)
Removes any white space from the right of %str and returns the result.

stripTrailingSpaces (*%str*)
Removes any spaces or underscores from %str and returns the result.

1.2 Processing

`strPos`(*%haystack*, *%needle*)
Searches for %needle in the %haystack *with* case-sensitivity. Returns the index of the beginning of %needle in %haystack on success and -1 when it cannot be found.

`striPos`(*%haystack*, *%needle*)
Searches for %needle in the %haystack *without* case-sensitivity. Returns the index of the beginning of %needle in %haystack on success and -1 when it cannot be found.

`strLen`(*%str*)
Returns the number of characters in %str.

`strCmp`(*%str1*, *%str2*)
Compares %str1 and %str2 **lexicographically**, i.e. in dictionary order, **with** case-sensitivity. Returns -1 if %str1 is less than %str2 (%str1 comes before %str2 in the dictionary), 0 if they are equal (%str1 and %str2 would be in the same place in the dictionary) and 1 if %str1 is greater than %str2 (%str1 comes after %str2 in the dictionary).

```
==> strCmp("bob","bane")
1
==> strCmp("bob","cat")
-1
==> strCmp("cat","cat")
0
```

`striCmp`(*%str1*, *%str2*)
Compares %str1 and %str2 **lexicographically**, i.e. in dictionary order, *without* case-sensitivity. Returns -1 if %str1 is less than %str2 (%str1 comes before %str2 in the dictionary), 0 if they are equal (%str1 and %str2 would be in the same place in the dictionary) and 1 if %str1 is greater than %str2 (%str1 comes after %str2 in the dictionary).

1.3 Words

Strings containing words, seperated by a space (" "), can also be processed and manipulated.

`firstWord`(*%str*)
Returns the first word in %str.

```
==> firstWord("Bob has a nice car.");
Bob
```

`restWords`(*%str*)
Returns every word in %str except for the first.

```
==> firstWord("Bob has a nice car.");
has a nice car.
```

`getWord`(*%str*, *%index*)
Returns the word at %index in %str.

```
==> getWord("Bob has a nice car.",1);
has
```

`getWordCount`(*%str*)
Returns the word count of %str.

`getWords`(*%str*, *%startIndex*[, *%endIndex*])
Returns all words in %str from %startIndex, or *optionally* all words from %startIndex to %endIndex.

removeWord(*%str*, *%index*)
Removes the word at %index in %str and returns the result.

setWord(*%str*, *%index*, *%word*)
Sets the the word in %str at %index to %word and returns the result.

1.4 Tokenizing

nextToken(*%string*, *%variableName*, *%delimeter*)
%string is the string containing tokens seperated by %delimiter. This function will split the string into seperate tokens (using %delimiter to split) and make a variable named %variableName in the scope with the next token's value.

```
function tokenExample(%string)
{
        %tokens = %string;
        while(%tokens !$= "")
        {
                %tokens = nextToken(%tokens,"token",":");
                echo(%token);
        }
}
==> tokenExample("Hello:this:is:being:tokenized");
Hello
this
is
being
tokenized
```

CHAPTER 2

Math

mAbs (*%n*)

Returns the absolute value (the distance from 0 on the number line) of %n.

```
==> mAbs(-5);
5
==> mAbs(5);
5
```

mAcos (*%n*)

A Trigonometrical function that returns the arc-cosine of %n, in radians.

mAsin (*%n*)

A Trigonometrical function that returns the arc-sine of %n, in radians.

mAtan (*%n*)

A Trigonometrical function that returns the arc-tangent of %n, in radians.

mCos (*%n*)

A Trigonometrical function that returns the cosine of %n.

mSin (*%n*)

A Trigonometrical function that returns the sine of %n.

mTan (*%n*)

A Trigonometrical function that returns the tangent of %n.

mCeil (*%n*)

Rounds %n up and returns the result.

mFloor (*%n*)

Rounds %n down and returns the result.

mClamp (*%n, %min, %max*)

Clamps %n between %min and %max. If %n is smaller than %min it returns %min, if %n is larger than %max it returns %max, otherwise it returns %n.

```
==> mClamp(5,1,10);
5
==> mClamp(-2,1,10);
1
==> mClamp(23,1,10);
10
```

mClampF (*%n, %min, %max*)

Clamps %n between %min and %max, supporting floating (decimal) point numbers. If %n is smaller than %min it returns %min, if %n is larger than %max it returns %max, otherwise it returns %n.

mDegToRad(*%n*)
Converts %n from radians to degrees and returns the result.

mRadToDeg(*%n*)
Converts %n from degrees to radians and returns the result.

mFloatLength(*%n*, *%length*)
Returns the floating (decimal) point value of %n with only %length number of decimal places with the last decimal place rounded.

```
==> mFloatLength(3.14159265,4);
3.1416
```

mLog(*%n*)
Returns the natural logarithm of %n.

mPow(*%n*, *%x*)
Returns %n to the power of %x.

```
==> mPow(2,2);
4
==> mPow(2,4);
16
==> mPow(16, 1 / 2);
4
```

mSqrt(*%n*)
Returns the square root of %n. This is the same as mPow(%n, 1 / 2);

CHAPTER 3

Vectors

Vectors are represented as 3 word strings. All vectors in TorqueScript follow the X Y Z format. X axis being left/right, Y being forward/back and Z being up/down.

vectorAdd(*%a*, *%b*)
Adds vector %a to vector %b and returns the result.

```
==> vectorAdd("-5 10 0","5 4 -2");
0 14 -2
```

vectorSub(*%a*, *%b*)
Subtracts vector %b from vector %a and returns the result. This is the same as adding the inverse of %b to %a.

vectorDot(*%a*, *%b*)
Performs a dot product between %a and %b and returns the result. This is the same as (%ax * %bx + %ay * %by + %az * %bz).

```
==> vectorDot("5 3 4","4 3 5");
49
==> 5 * 4 + 3 * 3 + 4 * 5
49
```

vectorCross(*%a*, *%b*)
Returns the cross product of vector %a and vector %b.

vectorDist(*%a*, *%b*)
Returns the distance between vector %a and vector %b. This is the same as (mSqrt(mPow(%bx - %ax,2) + mPow(%by - %ay,2) + mPow(%bz - %az,2)));

vectorLen(*%a*)
Returns the length/magnitude of vector %a. This is the same as (mSqrt(mPow(%ax,2) + mPow(%ay,2) + mPow(%az,2)));

vectorNormalize(*%a*)
Returns the normalized/unit vector of vector %a. This is the same as (%a / vectorLen(%a));

3.1 Examples

Getting the rotation between two points (by Truce):

```
function rotBetween(%a,%b)
{
        %v = vectorNormalize(vectorSub(%b,%a));
        %x = getWord(%v,0);
        %y = getWord(%v,1);
        %yaw = mATan(%x,%y) - $pi / 2;
        %pitch = 0 - mATan(getWord(%v,2),mSqrt(%x * %x + %y * %y));
        %xy = 0 - %pitch * 180 / $pi;
        %z = -90 - %yaw * 180 / $pi;
        return eulerToAxis(%xy SPC 0 SPC %z);
}
```

Popular third-party modifications:

CHAPTER 4

Return to Blockland

See the official RTB modding documentation written by the RTB developer, Ephialtes, for more information.

`RTB_registerPref`(*%prefName*, *%category*, *%variableName*, *%variableType*, *%addOnName*, *%defaultValue*, *%requiresRestart*, *%hostOnly*[, *%callback*])

Registers a preference that can be changed in the RTB Server Control interface. This should only be called while the server is starting up.

%prefName is the name of the preference that will be displayed in the Server Control menu.

%prefCategory is the category to display the preference in. Preferences with the same category are displayed together.

%variableName is the global variable name to link the RTB preference to. For example, *$Duplicator::Timeout*.

%variableType is the type of the variable that this preference will be.

- **"bool"** will only accept true or false values and will appear as a checkbox in the Server Control interface.
- **"list text value text value text value"** for a list. This will appear as a popup box in the Server Control interface. For each selectable option you want you need to include a text (the text to display in the selectable option) and the value (the value to set the preference to when the option is selected). For example, *"list North 0 South 1 East 2 West 3 Up 4 Down 6"*.
- **"int min max"** will accept an integer (whole number) between min and max and will appear as a textbox that auto corrects non-numbers. For example, *"int 1 10"* will result in a textbox that allows you to enter numbers between 1 and 10.
- **"string length"** will accept a string with a maximum character limit of length. It appears as a textbox.

%addOnName is the name of your add-on. It needs to be the file name without the .zip extension. For example, *"Tool_Duplicator"*.

%defaultValue is the default value for the preference. When the preference is first registered (say, if the user just downloaded) the preference is set to this value. Otherwise it is set to the saved value for the preference. This is also used when a user clicks the Defaults button in the Server Control interface.

%requiresRestart is a boolean value that will tell the user on the Server Control interface that they must restart the server for the changes to apply. It doesn't actually force the server to restart though.

%hostOnly is a boolean value that makes the preference only editable by the server host in the Server Control menu.

%callback is *optional*, it is the name of a function to call when the value of the preference is edited via the Server Control menu. The function is called with the old value as the first argument and the new value as the second argument.

```
    RTB_registerPref("Duplicator Timeout","Duplicator","$Duplicator::Timeout","int 0 60","Tool_Dupli
```

`RTB_registerGUI`(*%guiFile*)

Registers a GUI, %guiFile being the path to the GUI, with RTB. The GUI will be sent to all RTB users that join the server. This is essentially server-side GUIs. The restrictions on the client-side are very strict and RTB will block off any GUIs while they are downloading if they are found to be malicious. See the RTB modding documentation for more detailed information.

`RTB_addInfoTip`(*%tipMessage*, *%noBindTipMessage*, *%category*)

%tipMessage is the message to display when the tip is shown. It accepts Torque Markup Language formatting, as well as an additional (<key:keybindFunctionName>) markup tag to show the key mapping of a keybind.

%noBindTipMessage is the message to display when a keybind markup tag in %tipMessage doesn't have a binding. It also accepts Torque Markup Language formatting.

%category is un-used currently but in the future it will be used to group tips together.

4.1 Examples

The *correct* method for registering an RTB preference:

```
if(isFile("Add-Ons/System_ReturnToBlockland/server.cs"))
{
        if(!$RTB::RTBR_ServerControl_Hook)
        {
                //This sets the RTB preference system up. Don't remove this code!
                if(isFile("Add-Ons/System_ReturnToBlockland/hooks/serverControl.cs"))
                        exec("Add-Ons/System_ReturnToBlockland/hooks/serverControl.cs");
                else
                        exec("Add-Ons/System_ReturnToBlockland/RTBR_ServerControl_Hook.cs");
        }
        //Register your RTB prefs now and do any other RTB-based things.
}
else
{
        //The user doesn't have RTB so just set the global variables to their default value or do you
}
```

Using the RTB preference system functionality in a server command:

```
if(isFile("Add-Ons/System_ReturnToBlockland/server.cs"))
{
        if(!$RTB::RTBR_ServerControl_Hook)
        {
                if(isFile("Add-Ons/System_ReturnToBlockland/hooks/serverControl.cs"))
                        exec("Add-Ons/System_ReturnToBlockland/hooks/serverControl.cs");
                else
                        exec("Add-Ons/System_ReturnToBlockland/RTBR_ServerControl_Hook.cs");
        }
        RTB_registerPref("Enabled","Kill Mod","$Kill::Enabled","bool","Script_KillMod",1,0,0,"killMod
        RTB_registerPref("Host Only","Kill Mod","$Kill::HostOnly","bool","Script_KillMod",0,0,1);
        RTB_registerPref("Kill Message","Kill Mod","$Kill::Message","string 100","Script_KillMod","So
}
else
{
        $Kill::Enabled = 1;
        $Kill::HostOnly = 0;
        $Kill::Message = "Sorry %1, you were killed by %2.";
}
function serverCmdKill(%client,%targetName)
{
        if(!%client.isAdmin || !$Kill::Enabled || ($Kill::HostOnly && %client.bl_id != getNumKeyID())
                return;
        %targetClient = findClientByName(%targetName);
        %targetPlayer = %targetClient.player;
        if(isObject(%targetPlayer))
        {
                %message = strReplace($Kill::Message,"%1",%client.getPlayerName());
                %message = strReplace(%message,"%2",%targetClient.getPlayerName());
                messageClient(%targetClient,'',%message);
                %targetPlayer.kill();
        }
}
function killModToggled(%oldValue,%newValue)
{
        if(%newValue)
                echo("Kill mod is now enabled.");
        else
                echo("Kill mod is now disabled.");
}
```

CHAPTER 5

Indices and tables

- genindex
- search

F

firstWord() (built-in function), 4

G

getSubStr() (built-in function), 3
getWord() (built-in function), 4
getWordCount() (built-in function), 4
getWords() (built-in function), 4

L

ltrim() (built-in function), 3

M

mAbs() (built-in function), 7
mAcos() (built-in function), 7
mAsin() (built-in function), 7
mAtan() (built-in function), 7
mCeil() (built-in function), 7
mClamp() (built-in function), 7
mClampF() (built-in function), 7
mCos() (built-in function), 7
mDegToRad() (built-in function), 8
mFloatLength() (built-in function), 8
mFloor() (built-in function), 7
mLog() (built-in function), 8
mPow() (built-in function), 8
mRadToDeg() (built-in function), 8
mSin() (built-in function), 7
mSqrt() (built-in function), 8
mTan() (built-in function), 7

N

nextToken() (built-in function), 5

R

removeWord() (built-in function), 5
restWords() (built-in function), 4
RTB_addInfoTip() (built-in function), 12
RTB_registerGUI() (built-in function), 12
RTB_registerPref() (built-in function), 11
rtrim() (built-in function), 3

S

setWord() (built-in function), 5
strCmp() (built-in function), 4
striCmp() (built-in function), 4
stripChars() (built-in function), 3
stripMLControlChars() (built-in function), 3
striPos() (built-in function), 4
stripTrailingSpaces() (built-in function), 3
strLen() (built-in function), 4
strLwr() (built-in function), 3
strPos() (built-in function), 4
strReplace() (built-in function), 3
strStr() (built-in function), 3
strTrim() (built-in function), 3
strUpr() (built-in function), 3

T

trim() (built-in function), 3

V

vectorAdd() (built-in function), 9
vectorCross() (built-in function), 9
vectorDist() (built-in function), 9
vectorDot() (built-in function), 9
vectorLen() (built-in function), 9
vectorNormalize() (built-in function), 9
vectorSub() (built-in function), 9