Copyright © 2012 by Wavelink Corporation All rights reserved.

Wavelink Corporation
10808 South River Front Parkway, Suite 200
South Jordan, Utah 84095
Telephone: (801) 316-9000
Fax: (801) 316-9099
Email: customerservice@wavelink.com
Website: http://www.wavelink.com

No part of this publication may be reproduced or used in any form, or by any electrical or mechanical means, without permission in writing from Wavelink Corporation. This includes electronic or mechanical means, such as photocopying, recording, or information storage and retrieval systems. The material in this manual is subject to change without notice.

The software is provided strictly on an "as is" basis. All software, including firmware, furnished to the user is on a licensed basis. Wavelink grants to the user a non-transferable and non-exclusive license to use each software or firmware program delivered hereunder (licensed program). Except as noted below, such license may not be assigned, sublicensed, or otherwise transferred by the user without prior written consent of Wavelink. No right to copy a licensed program in whole or in part is granted, except as permitted under copyright law. The user shall not modify, merge, or incorporate any form or portion of a licensed program with other program material, create a derivative work from a licensed program, or use a licensed program in a network without written permission from Wavelink. The user agrees to maintain Wavelink's copyright notice on the licensed programs delivered hereunder, and to include the same on any authorized copies it makes, in whole or in part. The user agrees not to decompile, disassemble, decode, or reverse engineer any licensed program delivered to the user or any portion thereof.

Wavelink reserves the right to make changes to any software or product to improve reliability, function, or design.

The information in this document is bound by the terms of the end user license agreement.
# Table of Contents

Chapter 1: Introduction ........................................................................................................... 1

Chapter 2: Creating Scripts ................................................................................................. 3
  Launching the Script Editor ................................................................................................. 3
  Building Scripts Manually ................................................................................................. 5
  Selecting Activation Methods ............................................................................................ 5
    Select from Menu ............................................................................................................. 5
    Called from Another Script ............................................................................................. 5
    On Key Combination ........................................................................................................ 6
    When Session Connects .................................................................................................... 6
    On Barcode, MSR or RFID Scan ...................................................................................... 6
    On Screen Update ............................................................................................................. 8
  Building the Script Code ...................................................................................................... 9
    Verifying the Script Starts on the Correct Screen ......................................................... 9
    Entering the User Name and Password ......................................................................... 15
    Verifying the Screen and Navigating Menus ................................................................. 19
  Creating Variables ............................................................................................................. 25
  Selecting Host Profiles ...................................................................................................... 27
  Recording Scripts .............................................................................................................. 29
    Recording a Script ................................................................................................ .......... 29
    Adding a Field Data ID or Symbology .......................................................................... 31
  Creating Scripts From Text ............................................................................................... 34
    Importing a Text File ....................................................................................................... 34
    Building Scripts with the Text Editor ............................................................................ 34
    Calling Another Script .................................................................................................... 36

Chapter 3: Editing Scripts .................................................................................................... 38
  Turn Script Logging On ...................................................................................................... 39
  Turn Script Logging Off ..................................................................................................... 39

Chapter 4: Importing and Exporting Scripts ...................................................................... 40

Chapter 5: Organizing Scripts ............................................................................................. 42

Chapter 6: Executing Scripts ............................................................................................... 43
  Select From Menu ............................................................................................................. 43
  On Key Combination ......................................................................................................... 44
  When Session Connects ..................................................................................................... 44
  On Barcode, MSR, or RFID Scan .................................................................................... 45
  On Screen Update ............................................................................................................. 45
  From Web Pages ............................................................................................................... 45

Chapter 7: Overview of Actions .......................................................................................... 46
  Abort ................................................................................................................................. 62
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abort_All</td>
<td>63</td>
</tr>
<tr>
<td>Ask_Number</td>
<td>64</td>
</tr>
<tr>
<td>Ask_OK</td>
<td>66</td>
</tr>
<tr>
<td>Ask_OK_Cancel</td>
<td>67</td>
</tr>
<tr>
<td>Ask_String</td>
<td>68</td>
</tr>
<tr>
<td>Ask_String_Lowercase</td>
<td>69</td>
</tr>
<tr>
<td>Ask_String_Password</td>
<td>71</td>
</tr>
<tr>
<td>Ask_String_Uppercase</td>
<td>73</td>
</tr>
<tr>
<td>Ask_Yes_No</td>
<td>75</td>
</tr>
<tr>
<td>Ask_Yes_No_Cancel</td>
<td>76</td>
</tr>
<tr>
<td>Beep</td>
<td>77</td>
</tr>
<tr>
<td>Bitwise_And</td>
<td>78</td>
</tr>
<tr>
<td>Bitwise_Not</td>
<td>80</td>
</tr>
<tr>
<td>Bitwise_Or</td>
<td>82</td>
</tr>
<tr>
<td>Bitwise_Xor</td>
<td>84</td>
</tr>
<tr>
<td>Blank_Line</td>
<td>86</td>
</tr>
<tr>
<td>Boolean_And</td>
<td>87</td>
</tr>
<tr>
<td>Boolean_Equals</td>
<td>89</td>
</tr>
<tr>
<td>Boolean_Not</td>
<td>90</td>
</tr>
<tr>
<td>Boolean_Not_Equals</td>
<td>91</td>
</tr>
<tr>
<td>Boolean_Or</td>
<td>92</td>
</tr>
<tr>
<td>Boolean_Set</td>
<td>94</td>
</tr>
<tr>
<td>Break</td>
<td>95</td>
</tr>
<tr>
<td>Button_Bitmap_Create_Emulation</td>
<td>97</td>
</tr>
<tr>
<td>Button_Bitmap_Create_View</td>
<td>99</td>
</tr>
<tr>
<td>Button_Create_Emulation</td>
<td>101</td>
</tr>
<tr>
<td>Button_Create_View</td>
<td>102</td>
</tr>
<tr>
<td>Button_Remove</td>
<td>104</td>
</tr>
<tr>
<td>Button_Remove_All</td>
<td>105</td>
</tr>
<tr>
<td>Call</td>
<td>106</td>
</tr>
<tr>
<td>Cancel_Other_Scripts</td>
<td>110</td>
</tr>
<tr>
<td>Case</td>
<td>111</td>
</tr>
<tr>
<td>Character_To_Number</td>
<td>113</td>
</tr>
<tr>
<td>Comment</td>
<td>114</td>
</tr>
<tr>
<td>Continue</td>
<td>115</td>
</tr>
<tr>
<td>Default</td>
<td>116</td>
</tr>
<tr>
<td>Delay</td>
<td>117</td>
</tr>
<tr>
<td>Disconnect</td>
<td>118</td>
</tr>
<tr>
<td>Else</td>
<td>119</td>
</tr>
<tr>
<td>End_If</td>
<td>120</td>
</tr>
<tr>
<td>End_Switch</td>
<td>121</td>
</tr>
<tr>
<td>End_While</td>
<td>122</td>
</tr>
<tr>
<td>Escape_Sequence</td>
<td>123</td>
</tr>
<tr>
<td>Exit_Application</td>
<td>124</td>
</tr>
<tr>
<td>Message_Clear</td>
<td>179</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>Number_Divide</td>
<td>180</td>
</tr>
<tr>
<td>Number_Divide_Remainder</td>
<td>182</td>
</tr>
<tr>
<td>Number_Equal</td>
<td>184</td>
</tr>
<tr>
<td>Number_Greater_Than</td>
<td>185</td>
</tr>
<tr>
<td>Number_Greater_Than.Or_Equal</td>
<td>186</td>
</tr>
<tr>
<td>Number_Less_Than</td>
<td>187</td>
</tr>
<tr>
<td>Number_Less_Than.Or_Equal</td>
<td>188</td>
</tr>
<tr>
<td>Number_Minus</td>
<td>189</td>
</tr>
<tr>
<td>Number_Multiply</td>
<td>191</td>
</tr>
<tr>
<td>Number_Not_Equal</td>
<td>193</td>
</tr>
<tr>
<td>Number_Plus</td>
<td>194</td>
</tr>
<tr>
<td>Number_Set</td>
<td>196</td>
</tr>
<tr>
<td>Number_To_Character</td>
<td>198</td>
</tr>
<tr>
<td>Number_To_String_Binary</td>
<td>199</td>
</tr>
<tr>
<td>Number_To_String.Decimal</td>
<td>201</td>
</tr>
<tr>
<td>Number_To_String.Hexadecimal_Lowercase</td>
<td>203</td>
</tr>
<tr>
<td>Number_To_String.Hexadecimal_Uppercase</td>
<td>205</td>
</tr>
<tr>
<td>Number_To_String.Octal</td>
<td>207</td>
</tr>
<tr>
<td>Play_Sound</td>
<td>209</td>
</tr>
<tr>
<td>Printer_Cancel</td>
<td>210</td>
</tr>
<tr>
<td>Printer_Data</td>
<td>211</td>
</tr>
<tr>
<td>Printer_Repeat</td>
<td>213</td>
</tr>
<tr>
<td>Reboot</td>
<td>215</td>
</tr>
<tr>
<td>Return</td>
<td>216</td>
</tr>
<tr>
<td>Run_Application</td>
<td>217</td>
</tr>
<tr>
<td>Scan_String</td>
<td>218</td>
</tr>
<tr>
<td>Search_Screen</td>
<td>219</td>
</tr>
<tr>
<td>Set_Cursor_Position</td>
<td>221</td>
</tr>
<tr>
<td>Set_Field_Append.Scan_Data</td>
<td>222</td>
</tr>
<tr>
<td>Set_Field_Com_Data_Field</td>
<td>223</td>
</tr>
<tr>
<td>Set_Field_Data_ID</td>
<td>224</td>
</tr>
<tr>
<td>Set_Field_Prefix.Scan_Data</td>
<td>225</td>
</tr>
<tr>
<td>Set_Field_Symbology_ID</td>
<td>226</td>
</tr>
<tr>
<td>Speech.Change.Setting</td>
<td>228</td>
</tr>
<tr>
<td>Speech.Find.Setting.Value</td>
<td>230</td>
</tr>
<tr>
<td>Speech.From_Text</td>
<td>231</td>
</tr>
<tr>
<td>Speech.From_Text.Available</td>
<td>233</td>
</tr>
<tr>
<td>Speech.From_Text.Cancel</td>
<td>234</td>
</tr>
<tr>
<td>Speech.From_Text.Error.Desc</td>
<td>235</td>
</tr>
<tr>
<td>Speech.Get.Confidence.Level</td>
<td>236</td>
</tr>
<tr>
<td>Speech.Get.Setting</td>
<td>237</td>
</tr>
<tr>
<td>Speech.Get.Setting.Max</td>
<td>238</td>
</tr>
<tr>
<td>Speech.Get.Setting.Value_Desc</td>
<td>239</td>
</tr>
</tbody>
</table>
Speech_Setting_Available ................................................................. 240
Speech_To_Text ................................................................. 241
Speech_To_Text_Available ............................................................... 242
Speech_To_Text_Cancel ................................................................. 243
Speech_To_Text_Change_User_Name ...................................................... 244
Speech_To_Text_Error_Desc ............................................................. 245
Speech_To_Text_Get_User_Name .......................................................... 246
Speech_To_Text_No.Wait ................................................................. 247
String_Combine ................................................................. 251
String.Empty ................................................................. 252
String.Equal ................................................................. 253
String.Find_First ................................................................. 255
String.Find_Last ................................................................. 257
String.Greater_Than ................................................................. 259
String.Greater_Than_Or_Equal ......................................................... 261
String.Left ................................................................. 263
String.Length ................................................................. 264
String.Less_Than ................................................................. 265
String.Less_Than_Or_Equal ............................................................. 267
String.Lower ................................................................. 269
String.Middle ................................................................. 270
String.Not_Equal ................................................................. 272
String.Only_Characters ............................................................... 274
String.Replace ................................................................. 276
String.Right ................................................................. 277
String.Set ................................................................. 278
String.Strip_Characters .............................................................. 279
String.To_Number_Binary .............................................................. 281
String.To_Number.Decimal ............................................................. 283
String.To_Number.Hexadecimal ......................................................... 285
String.To_Number.Octal ................................................................. 287
String.Trim_Spaces_End ............................................................... 289
String.Trim_Spaces_Start ............................................................... 290
String.Upper ................................................................. 291
Suspend ................................................................. 292
Switch ................................................................. 293
Wait_For_Screen_Update ............................................................... 295
Wait_For_Screen_Update_With_Timeout .................................................. 296
Web_Get_Current_Element .............................................................. 297
Web_Get_Source ................................................................. 298
Web_Navigate ................................................................. 299
Web_Navigate_Frame ................................................................. 300
Web_Navigate_Post_Data ............................................................... 301
Web_Scripting ................................................................. 302
Chapter 8: Speakeasy Settings

tts_calibrate ................................................................. 313
tts_externalSpeaker_setting ........................................... 314
tts_frequency ................................................................. 315
tts_language_long .......................................................... 316
tts_language_short .......................................................... 318
tts_pitch ................................................................. 322
tts_rate ................................................................. 323
tts_readmode ................................................................. 325
tts_voice ................................................................. 326
tts_volume ................................................................. 327
tts_waitfactor ................................................................. 330
stt_accuracy ................................................................. 332
stt_adjust_gain ............................................................... 333
stt_beep_threshold .......................................................... 334
stt_calibrate ................................................................. 335
stt_calibration_silence ..................................................... 336
stt_confidence ................................................................. 337
stt_expanded ................................................................. 338
stt_fx_detect_start ........................................................... 339
stt_fx_microphone ........................................................... 340
stt_fx_min_duration .......................................................... 341
stt_fx_sensitivity ............................................................. 342
stt_fx_silence ................................................................. 343
stt_fx_threshold ............................................................... 344
stt_idle_timeout .............................................................. 345
stt_language_long ............................................................ 346
stt_language_short ............................................................ 347
stt_logging ................................................................. 348
stt_logging_audio ............................................................. 349
stt_logging_engine ............................................................ 350
stt_pool_size ................................................................. 351
stt_preserve ................................................................. 352
stt_priority ................................................................. 353
stt_processing ................................................................. 354
stt_reset ................................................................. 355
stt_reset_session_delay ....................................................... 356
stt_result_sound ............................................................... 357
stt_save_increase ............................................................. 358
stt_save_session_delay ........................................................ 359
Chapter 1: Introduction

This document provides information about creating and executing scripts for the Wavelink Terminal Emulation (TE) Client using the TE Script Editor.

The Script Editor is a component of the Wavelink TE Client. The Script Editor provides the ability to create and execute scripts that automate processes on the TE Client. The information in this document is for 7.3.5.9 and later builds of the TE Client.

The following steps outline the process of creating scripts using the Script Editor:

1  **Launch the Script Editor.** Launch the Script Editor from the TE Client installation utility or from the Avalanche Console.

2  **Create scripts using the Script Editor.** Use the Script Editor to manually build the script code.

   -Or-

   **Create scripts using the Script Capture option.** Capture the actions you want to include in your script to build the script code.

   -Or-

   **Create scripts from text.** Import a text file or create a text-based script using the Text Editor.

3  **Configure an execution method for the script.** Select from the available options the method you want to use to execute your script.

4  **Deploy the script to the TE Client.** Using Wavelink Avalanche, ActiveSync, or a third-party application, deploy the script to the devices.

5  **Execute the script from the TE Client.** Using the activation method you selected for the script, activate and execute the script.

**NOTE:** You can only run scripts while a Terminal Emulation session is active. If the connection drops, the script is terminated. If you switch between sessions, the script running in the inactive session remains suspended until that session is active again.

This document is written with the assumption the reader and user of the TE Client possess:

- Knowledge of wireless networks and wireless networking protocols.
- Knowledge of TCP/IP.
- Knowledge of Wavelink Terminal Emulation.
- Knowledge or rudimentary experience with programming/scripting languages.
The following table lists the document conventions used in this manual.

<table>
<thead>
<tr>
<th>Text Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Courier New</strong></td>
<td>Any time you type specific information (such as a file name) or press keys, those options appear in the <strong>Courier New</strong> text style. This text style is also used for sample code. Example: Press CTRL+ALT+DELETE.</td>
</tr>
<tr>
<td><em>Italic Courier New</em></td>
<td>Parameters are represented in italic Courier New font. The descriptions of the parameters, when necessary, are listed at the right.</td>
</tr>
<tr>
<td><strong>Bold</strong></td>
<td>Any time you interact with an option (such as a button or descriptions of different options in a dialog box), that option appears in the <strong>Bold</strong> text style. Examples: Click <strong>File &gt; Open</strong>. Click the <strong>Update</strong> button.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Titles of dialog boxes are represented with an <em>italic</em> text style. Example: The <em>Script Editor</em> dialog box appears.</td>
</tr>
</tbody>
</table>

**NOTE:** If you copy and paste the sample code, you may need to edit out returns.

For concision and clarity, the name "Avalanche" is used to refer to both Avalanche MC and Avalanche SE. For more information about each product, refer to the specific user guide.
Chapter 2: Creating Scripts

There are three ways you can create scripts:

- **Manually.** Using this method, you build the script code from scratch using the Script Editor.

- **Script Capturing.** Using this method, you generate the script code by enabling script capturing and performing the action you want the script to automate. The Script Editor records the key presses and cursor movements and saves the information as script code. You can edit the captured code to customize the way the script runs.

- **From Text.** Using this method, you generate the script code by importing text from any text editor or entering text in the Script Editor.

For testing scripts, Wavelink recommends you install the TE Client for Windows. Once the script has been refined, you can save and deploy the script to the mobile devices you want to run the script.

This section provides information about creating scripts, including:

- Launching the Script Editor
- Building Scripts Manually
- Recording Scripts
- Creating Scripts From Text
- Calling Another Script

**Launching the Script Editor**

The Script Editor can be launched from the TE Client for Windows, the Configuration Utility for an ActiveSync installation, or from the Avalanche Console. Wavelink recommends creating and performing initial testing using the TE Client for Windows, then importing the scripts using either the ActiveSync utility or the Avalanche Console.

**To launch the Script Editor from the TE Client:**

1. Click **Options > Configure > Script Editor.**

   The Script Editor appears.

2. Click **Add** to access the **Script Editor Configuration** dialog box.

**To launch the Script Editor for a Client installed via ActiveSync:**

1. From the Wavelink Product Configuration Utility, click **Script Editor.**
The Script Editor appears.

2 Click **Add** to access the *Script Editor Configuration* dialog box.

**To launch the Script Editor from Avalanche Console:**

1 Ensure that the TE Client package is installed in the Avalanche Console.

2 Launch the Avalanche Console.

3 In the **Software Profiles** tab, locate the profile that contains the TE Client software package.

4 In the **Software Packages** tab, select the TE Client software package and click **Configure**.

   The *Configure Package* dialog box appears.

5 From the menu list, select **Script Editor** and click **OK**.

   The Script Editor appears.

6 Click **Add** to access the *Script Editor Configuration* dialog box.

---

**Script Editor Configuration Dialog Box**

From this dialog box, you can configure and create scripts. You must create a name for the script before you can begin editing it.
Building Scripts Manually

When building a script manually, you build the code for the script line by line in the Script Editor. The Script Editor allows you to name the script, select an activation method, use prompts to write the code, create variables, and select the host profiles to associate with the script.

This section provides the following information:

- Selecting Activation Methods
- Building the Script Code
- Creating Variables
- Selecting Host Profiles

Selecting Activation Methods

The activation method determines the way you execute the script from the TE Client. A script with no activation method selected can only be called by another script. It cannot activate alone.

Activation methods are selected in the General Settings tab of the Script Editor Configuration dialog box. This section provides information about the different activation methods including:

- Select from Menu
- Called from Another Script
- On Key Combination
- When Session Connects
- On Barcode, MSR or RFID Scan
- On Screen Update

Select from Menu

The Select from Menu option places a script execution selection in the TE Client Options or Term menu.

Called from Another Script

The Called from Another Script option allows the script to be called from another script. This option is automatically enabled when a script uses parameters.
On Key Combination

If you enable the On Key Combination option, you can launch the script by pressing the specified keys.

Use the Diagnostic Utility to obtain the key value. For more information about using the Diagnostic Utility, see the Wavelink Terminal Emulation Client User Guide.

When Session Connects

If you enable the When Sessions Connects option, the script activates when the host profile it supports is activated.

If you use this option, Wavelink strongly recommends that you limit the script to the appropriate host profiles. Because the script activates before any information appears on the emulation screen, you need to have your script wait for the appropriate screen to appear before activates. You should not have more than one script set to start when a session connects because the first script that starts will prevent any other scripts from running while it waits for the initial screen. For more information, see Selecting Host Profiles on page 27.

On Barcode, MSR or RFID Scan

If you want to perform special processing on items scanned into the computer, the Scan Handler is often powerful enough to make the changes you need. The Scan Handler settings, found in the Configuration Manager, are located in Emulation Parameters > Scanner > Common > Scan Handler. However if the Scan Handler is insufficient, you can use a script to perform any processing you need.

Before you can activate the script for a scan, you must create a string variable and a number variable.

- The string variable allows you to obtain the initial scan data.
- The number variable allows you to obtain the type of scan data.

Refer to Symbologies and Values on page 369 for the values of different symbologies. You can also use the Get Scan Type Name and Get Scan Type Value commands to display or handle scan types. Using the Get Scan Type Value means all types are specified in the editor and you can select the type you want to use.

Calling the Scan String command before your script exits allows Terminal Emulation to handle the scanning data. Because you are specifying the data and type returned, the script can change either one. If the script exits without calling Scan String, the scanned data disappears.

To configure the On Barcode, MSR, or RFID Scan method:

1. Create the Scan String and Scan Type variables.
Once you create these variables, the On Barcode, MSR or RFID Scan option becomes available.

Create these variables in the String Variables and Number Variables tabs. For information on variables, see Creating Variables on page 25.

2 Select the General tab or Activate tab in the Script Editor.

3 Enable the On Barcode, MSR or RFID Scan option.

4 From the drop-down menu, select the Scan_String.

5 From the drop-down menu select the Scan_Type.

6 Click OK.

On Barcode, MSR or RFID Scan Example 1

The following is a sample script you can use if you want to insert a string (which could be just one character long) after the first six characters of any barcode at least six characters long.

A few notes about the sample script:

- **ScanData** is a string variable with the original barcode.
- **NewString** is a variable where you store the new barcode.
- **ScanType** is the number variable that keeps the type of scan data received.
- **OldLength** is an integer variable.
- **XXYY** is the string you insert.

```plaintext
OldLength = String_Length(ScanData)
  If (Number_Greater_Than_or_Equal(OldLength, 6))
NewString = String_Combine(String_Left(ScanData, 6), "XXYY")
  NewString = String_Combine(NewString, String_Right(ScanData, Number_Minus(OldLength, 6)))
Else
  NewString = ScanData
End_If
Scan_String(NewString, ScanType)
Return
```

On Barcode, MSR or RFID Scan Example 2

This example converts any DataMatrix scan values to PDF417 scan values. The ScanData and ScanType variables described for the previous example are used again.

```plaintext
If (Number_Equal(ScanType, Get_Scan_Type_Value("DATAMATRIX")))
Scan_String(ScanData, Get_Scan_Type_Value("PDF417"))
Else
```
Scan_String(ScanData,ScanType)
End_If
Return

On Screen Update

This option activates the script (if activation is allowed) every time the text on the emulation screen changes. This includes updates from the host or when the user presses a key and the key value is shown on the screen. It is recommended that you limit the host profiles that the script supports.

The following example generates a script that enters a command each time a particular string appears on the screen:

Label:Start:
If (String_Equal(Get_Screen_Text_Columns(1,1,5),"Ready", 0,FALSE))
Keypress_String("Proceed")
Keypress_Key("Enter")
End_If
Wait_For_Screen_Update
Goto: Start
Return

If the script is set to activate when the session first connects, it will work as desired with one limitation. Because it is always activated, no other scripts can be activated during the emulation session.

Here is an alternate implementation:

If (String_Equal(Get_Screen_Text_Columns(1,1,5), "Ready", 0, FALSE))
Keypress_String("Proceed")
Keypress_Key("Enter")
End_If
Return

If the script is set to run each time the screen updates, you get the desired behavior. Because the script is not active all the time, other scripts can be activated as well.

**NOTE:** Use this option carefully as it can cause a script to be executed frequently.

To configure the On Screen Update method:

1. Select the General tab or Activate tab in the Script Editor.
2. Enable the On Screen Update option.
3. Click OK.
Building the Script Code

Once you name the script and select an activation method, you can begin entering the code. You build the code in the **Actions** tab.

The sections below use an example script to demonstrate how to create the following script actions:

- Select a specific menu and then exit the script.
- **Verifying the Script Starts on the Correct Screen** and setting the correct cursor position. If the script is not on the correct screen, the script ends. If the emulation screen is correct, the script enters a user name and password in the correct locations.
- **Entering the User Name and Password** to send to the host.
- **Verifying the Screen and Navigating Menus** during a connection session.

**Verifying the Script Starts on the Correct Screen**

This portion of script verifies that the script is starting on the right screen and sets the correct cursor position. If the script is not on the right screen, the script ends. Once the script verifies that it is starting on the correct screen, a message displays "Starting Script."

**To build example verification script:**

1. From the **Actions** tab, click the **Insert** button.
   
   The *Action Editor* dialog box opens.

2. From the **Action** drop-down menu, select **Comment**.

3. Click the **Comment** tab and enter **Verify that this is the desired screen in the Constant String** text box.

4. Click **OK**.

   The code is added to the **Actions** tab.
Enter a Comment

5  Click the **Insert** button.
6  From the **Action** drop-down menu, select **If_Not**.
7  Click the **Test** tab.
8  Enable the **Action** drop-down option and select **String_Equal** from the drop-down menu.
9  Click the **Edit Action Value** button.
10 Click the **Test 1** tab.
11 Enable the **Action** drop-down option and select **Get_Screen_Test_Length** from the drop-down menu.
12 Click the **Edit Action Value** button.
13 Click the **Row** tab and enter the number 1.
14 Click the **Column** tab and enter the number 38.
15 Click the **Maximum Length** tab and enter the number 7.
16 Click **OK**.
17 Click the **Test 2** tab and enter **Sign On** in the **Constant String** text box.
18 Click the **Maximum Length** tab and enter the number 0 in the **Constant Number** text box.

19 Click the **Ignore Case** tab and enable the **False** option.

20 Click **OK** until you return to the **Actions** tab in the Script Editor.

   The code appears in the **Actions** tab.

![Script Editor](image)

---

**Entering an If_Not Action**

21 Click the **Insert** button.

22 From the **Action** drop-down menu, select **Return**.

23 Click **OK**.

   The code appears in the **Actions** tab.
Chapter 2: Creating Scripts

24 Click the Insert button.

25 From the Action drop-down menu, select End If.

26 Click OK.

The code appears in the Actions tab.

---

**Entering a Return**

1 Comment: Verify that this is the desired screen.
2 If Not String_Equal(Get_Screen_Text_Len(2, 48, 6), "System", 0, FALSE) Then
3 Return
4 Return
27. Click the **Insert** button.

28. From the **Action** drop-down menu, select **Set_Cursor_Position**.

29. Click the **Row** tab and enter 6 in the **Constant Number** text box.

30. Click the **Column** tab and enter 53 in the **Constant Number** text box.

31. Click **OK**.

The code appears in the **Actions** tab.
Chapter 2: Creating Scripts

32 From the **Action** tab, click the **Insert Blank** button to insert a blank line in the code.

33 Click the **Insert** button.

34 From the **Action** drop-down menu, select **Message**.

35 Click the **Message** tab and enter **Starting Script** in the **Constant Number** text box.

   This code displays a "Starting Script" message on the emulation screen.

36 Click the **Timeout (Seconds)** tab and enter the number 3 in the **Constant Number** text box.

37 Click **OK**.

   The code appears in the **Actions** tab.
Chapter 2: Creating Scripts

Entering the Message Code

Entering the User Name and Password
This portion of the script enters the login information.

To build the example script:
1 Click the **Insert Blank** button to insert a blank line in the code.
2 Click the **Insert** button.
3 From the **Action** drop-down menu, select **Keypress_String**.
4 Click the **Characters** tab and enter *User Name* in the **Constant String** text box.
5 Click **OK**.

The code appears in the **Actions** tab.
Chapter 2: Creating Scripts

Entering the User Name Code

6 Click the Insert button.

7 From the Action drop-down menu, select Keypress_Key.

8 Click the Key tab.

9 From the Emulation drop-down menu, select the emulation type.

10 From the Key drop-down menu, select Down Arrow.

11 Click OK.

The code appears in the Actions tab.
12 Click the Insert button.

13 From the Action drop-down menu, select Keypress String.

14 Click the Character tab and enter Password in the Constant String text box.

15 Click OK.

The code appears in the Actions tab.
Chapter 2: Creating Scripts

16 Click the **Insert** button.

17 From the **Action** drop-down menu, select **Keypress Key**.

18 Click the **Key** tab.

19 From the **Emulation** drop-down menu, select your emulation type.

20 From the **Key** drop-down menu, select **Enter**.

21 Click **OK**.

The code appears in the **Actions** tab.
Verifying the Screen and Navigating Menus

This portion of the code verifies that you are on the correct screen after login. If you are not on the correct screen, the script will wait until the correct screen appears. Once the you are on the correct screen, the script navigates to a specific menu. The script displays the message "Script Done" and the script exits.

To build the example verification script:

1  Click the Insert Blank button.

2  Click the Insert button.

3  From the Action drop-down menu, select Comment.

4  Click the Comment tab and enter Wait for desired screen in the Constant String text box.

5  Click OK.

The code appears in the Actions tab.
6. Click Insert.

7. From the Action drop-down menu, select While_Not.

8. Click the Test tab.

9. From the Action drop-down menu, select String_Equal.

10. Click the Edit Action Value button.

11. Click the Test 1 tab.

12. From the Action drop-down menu, select Get_String_Text_Length.

13. Click the Test 2 tab and enter 9. FUNCTION KEYS in the Constant String text box.

14. Click the Maximum Length tab and enter the number 0 in the Constant Number text box.

15. Click the Ignore Case tab and enable the FALSE option.

16. Click the Test 1 tab.

17. Click the Edit Action Value button.

18. Click the Row tab and enter the number 11 in the Constant Number text box.
19 Click the **Column** tab and enter the number 1 in the **Constant Number** text box.

20 Click the **Maximum Length** tab and enter the number 16 in the **Constant Number** text box.

21 Click **OK** until you return to the **Action** tab.

The code is added to the **Action** tab.

```
2  If_Not( String_Equals( Get_Screen_Text_Length( 1, 36, 7 ), "Sign On", 0, FALSE ) )
3    Return
4  End_If
5  Set_Cursor_Position( 6, 53 )
6  Message( "Starting Script!", 3 )
7  Keypress_String( "User Name" )
8  Keypress_Key( "Down Arrow" )
9  Keypress_String( "Password" )
10  Keypress_Key( "Enter" )
11  Comment: Wait for the desired screen
12
13  While_Not( String_Equals( Get_Screen_Text_Length( 11, 1, 16 ), "9. FUNCTION KEYS", 0, FALSE ) )
14    Return
15
16
```

22 Click the **Insert** button.

23 From the **Action** drop-down menu, select **Wait_For_Screen_Update**.

24 Click **OK**.

The code appears in the **Actions** tab.
Chapter 2: Creating Scripts

Entering a Wait_For_Screen_Update Action

25 Click the **Insert** button.

26 From the **Action** drop-down menu, select **End_While**.

27 Click **OK**.

The code appears in the **Actions** tab.
28 Click the **Insert Blank** button to insert a blank line in the code.

29 Click the **Insert** button.

30 From the **Action** drop-down menu, select **Keypress_String**.

31 Click the **Characters** tab and enter the number 9 in the **Constant String** text box.

32 Click **OK**.

The code is added to the **Actions** tab.
Chapter 2: Creating Scripts

33 Click the **Insert Blank** to insert a blank line in the code.

34 Click the **Insert** button.

35 From the **Action** drop-down menu, select **Message**.

36 Click the **Message** tab and enter **Script Done** in the **Constant String** text box.

37 Click the **Time(Milliseconds)** tab and enter the number **3** in the **Constant Number** text box.

38 Click **OK**.

The code is added to the **Actions** tab.
Chapter 2: Creating Scripts

Entering a Message Action

The code is complete.

39 Click OK to save the code in the Script Editor script list.

Once you complete the script you can deploy it to a device and execute it.

Creating Variables

There are three types of values recognized by scripting:

- Booleans (TRUE or FALSE values only)
- Numbers (integers)
- Strings

Every argument for every action is one of these three value types. Every action that returns a value returns one of these types. Variables provide a way to save the result of an action for later use as an argument for another command.

You can create and edit variables located in the corresponding tabs while editing a script. You can also create new variables while editing an action.
When a variable is persistent, the value remains after the script exits. Persistent variables are not script or session specific; once a value is assigned, any script in any emulation session can use a persistent variable to access that value. Two scripts are considered to be referencing the same persistent variable if both scripts contain persistent variables of the same type and same name.

Because writing new values to persistent variables will slow your application, they should only be used when necessary. If you want to use a persistent variable that will change values frequently, write your script with a regular variable that only changes the value of the persistent variable before the script pauses or exits.

**NOTE:** When you are using scripting with Speakeasy, any number variables need to be converted to string variables before text-to-speech can read them.

When a script starts, the variables will have known values: boolean variables are FALSE, number variables are 0, and string variables are empty. Possible exceptions to this are persistent variables or when a script activates another script. For more information on using a script to activate another script, see Calling Another Script on page 36.

**To create variables:**

1. Determine which type of variable you want to create: boolean, number, or string.
2. From the Script Editor, select the tab that corresponds with the type of variable you want to create.
3. Click **Add**.
4. In the *Edit Variable* dialog box, enter the name of the new variable.

![Adding a New Variable](image)

5. If you would like the variable to be a persistent variable, enable the **Make this variable persistent** checkbox.
6. Click **OK**.

The new variable appears in the corresponding tab.
New Variable

Selecting Host Profiles

For each script, you can specify which host profiles will be supported by that script. You may select host profiles from the Host Profiles tab.

If the script is generated by script capturing, limit the script to the host profile that was in use when the script was captured. The default — no host profile — allows the script to be run when any host profile is used.

To select host profiles:

1. From the Script Editor, select the Host Profiles tab.
Host Profiles Tab

2 Click **Add**.

The *Select Host* dialog box opens.

Selecting Host Profiles

3 Select which host you want to use from the list of Avalanche hosts.
Chapter 2: Creating Scripts

NOTE: If you have not created any host profiles, this dialog box will be empty.

4   Click OK.

The host appears in the Host Profiles tab.

Recording Scripts

Script capturing is an easy way to generate a script that automates actions or processes. When script capturing is activated, it captures key presses and mouse/pen cursor movements so those actions can be replayed when the script is activated. The script is saved to a file and can be edited using the Script Editor or a Text Editor after the script has been captured.

In addition to key presses and cursor movements, you can set field data identifiers or symbology values while you are recording a script.

- Recording a Script
- Adding a Field Data ID or Symbology

Recording a Script

When script capturing is activated, it captures key presses and mouse/pen cursor movements so those actions can be replayed when the script is activated. You can record a script from a device running a TE Client or using the TE Client for Windows.

To perform a script capture:

1   Position your mouse or cursor on the emulation screen you want to be at when the automated process starts.

2   From the Options or Term menu, select Scripting > Start Capture.

   The Script Capturing Initialization dialog box appears, asking if you want to verify the current screen text.

3   Select Yes to verify the current screen text.

   Select No if you do not want to verify the current screen text.

   Selecting Yes makes the captured script start with an If_not command that tells the script to exit if the correct screen is not currently shown. Unless you know that your script will only run from the correct screen (for example, a script that is run only when a session first starts, or a script called by another script), you should select Yes.
NOTE: If you select No, click Verify Screen Contents and Save Cursor Position when you start your script capture. This causes your script to wait for the Client to finish updating the screen before processing script actions.

4 Perform any actions you want to include in the script.

5 Each time the screen changes, click Verify Screen Contents button.

NOTE: Some devices may only display buttons labeled Screen, Cursor and Stop. The Screen button refers to the Verify Screen Contents button. The Cursor button refers to the Save Cursor Position button. The Stop button refers to the Stop Capturing button.

6 When you finish capturing the behaviors you want in the script, click Stop Capture.

Once you capture a script, the Script Editor opens, allowing you to name the script and select an activation method. You can use the Actions tab to add actions for any error condition that the user may encounter.
Adding a Field Data ID or Symbology

The field data ID feature allows you to use scripting to configure field data identifiers. Field data identifiers assign a unique identification, such as a letter, to each field on the screen. Any time a barcode beginning with that identifier is scanned, the information automatically populates in the corresponding field.

In addition to field data IDs, you can use the barcode symbology to determine the field where scanned data should go.

**NOTE:** This feature is only available for IBM 5250 emulation.

Use the *Field Data ID* dialog box to add a data ID or symbology to a field. If you want to make changes to data ID or symbology that has already been added to a script, you must edit the script manually.

**To add a Field Data ID or Symbology:**

1. Connect to an IBM 5250 host.

2. From the Term menu, select **Scripting > Start Capture**.

   The *Script Capturing Initialization* dialog box appears.

3. Click **Yes**.

   The *Select Screen Text* dialog box appears.

4. Select the desired text string or strings from the list and click **OK**.

5. Place the cursor on the field where you want to add a Data ID.

6. From the bottom of the emulation screen, click the **Field Data ID** button.
Field Data ID Button

The *Field Data ID* dialog box appears.
7 In the text box below the Add Data ID button, enter the desired field data ID.

8 Click Add Data ID.

The new Data ID appears in the list below the text box.

9 If you want to remove a data ID, select the desired ID from the list and click Remove.

10 From the drop-down menu below the Add Symbology button, select the desired symbology.

11 Click Add Symbology.

The new symbology appears in the list below the text box.

12 If you want to remove a symbology, select the desired symbology from the list and click Remove.

13 If you want to add a prefix to the data, enable the Prefix Data checkbox and enter the prefix in the available text box.

- If you want to append scan data in the field, enable the Append Scan Data checkbox.
- If you want to add a data ID and a symbology, enable the Data AND Symbology checkbox.
• If you want the field to be the Com Data Field for the screen, enable the **Com Data Field** checkbox.

14 Click **OK** to close the *Field Data ID* dialog box and save your changes.

## Creating Scripts From Text

You can use the Script Editor to generate scripts from standard text. The text can be entered in the Script Editor’s built-in text editor, or imported from a text-editing program such as Notepad. This section provides information about creating scripts from text, including the following:

- Importing a Text File
- Building Scripts with the Text Editor

### Importing a Text File

Create a script in any standard text editor and then import the script into the Text Editor. When creating the script text, you must use the same actions and commands provided in the Script Editor.

**To import a text file:**

1. Launch the Script Editor.
2. Click **Import Text**.

   The *Select the Script Text File* dialog box appears.

3. Navigate to and select the text file.
4. Click **Open**.

   The file is imported into the Script Editor.

### Building Scripts with the Text Editor

You can also create a text-based script using the Text Editor. The Text Editor will validate the script syntax when you build the script.

**To build a script:**

1. Launch the Script Editor.
2. Click **Add As Text**.

   The Text Editor opens.
3  Enter the text into the Text Editor. You must include a script name, along with script variables and activation methods. If desired, you may also assign a Host Profile to be supported by the script.

4  After entering your text, click Build to verify the script text. The Text Editor displays build information in the bottom section of the window. If the build information includes an error message, single-click the message to display the Script Editor Error Help dialog box.

The Script Editor Error Help dialog box displays the error number, a description of the error, and explains how to fix the error.
### Script Editor Error Help

For a full description of each error message, refer to [Script Build Errors](#) on page 377.

5. When you have finished building the script, click **Save** or **Save As** to save a copy of the script text.

6. Click **Done** to add the script to the Script Editor, or click **New** to begin building a new script.

### Calling Another Script

You can program a script to call other scripts or to call itself. Nesting scripts makes it easier to take a block of functionality and use it multiple times or to solve problems that can be described recursively.

The following example script shows how to use recursion (a script calling itself) to calculate factorials:

```plaintext
If (Number_Equal(ArgumentValue,1))
Comment: The factorial of 1 is 1
Return
End_If
If (Number_Not_Equal(ArgumentValue,0))
Comment: The factorial of X is X multiplied by the factorial of X - 1
Temp=ArgumentValue
ArgumentValue=Number_Minus(Temp,1)
Call: Factorial
ArgumentValue <-> ArgumentValue
ArgumentValue = Number_Multiply(Temp,ArgumentValue)
Return
End_If
```
ArgumentValue = Ask_Number("Enter a number:", "Factorial Calculator",1,12,0)
Call: Factorial
ArgumentValue <-> ArgumentValue
Ask_OK(String_Combine("The factorial is", Number_To_String_Decimal(ArgumentValue)), "Result")
Return

This script uses two integer variables, ArgumentValue and Temp.

When a script calls another script, the calling script can assign values to the called script variables. The factorial example script knows it is being called recursively because the ArgumentValue variable is not 0. If ArgumentValue is 0, the script will ask for the number to calculate the factorial with. Each time the script is called, the ArgumentValue variable of the calling script is assigned the final value in the called script’s ArgumentValue variable. This keeps the results of the called script’s actions from being lost. (If the value were not returned, then there would be a "<--" instead of a "<->" in the Call action’s argument list.) When you add the action for calling a script, you need to specify which variables in the called script will be assigned a value, and what that initial value will be.

**NOTE:** If you wanted to be more efficient, you could create a While loop that performs the multiplications to calculate the factorial. You could also return the proper response for each factorial, since numbers higher than 12 exceed the maximum number value. However, there are some problems that are easiest to solve using recursion. The example above may give you an idea how you could use it.
Chapter 3: Editing Scripts

Regardless of how a script is created, edit it using either the Script Editor or a text editor. You can debug a TE Client script from the TE Text Editor or have the script generate a log file. Once you have completed editing the script you can export the script to use on a different device or deploy it for use during an emulation session.

When the script generates a log file, each action executed, with the values of its arguments and the results of the action, is written to the log file. The file path for where the log file is stored is configured using the Logging_On action.

When a script calls another script, the logging for the calling script is suspended. Logging resumes when the called script exits. It is possible to have a script called by another script (or a script calling itself recursively) use the same logging file.

For more information on using script logging, see the following sections:

- Turn Script Logging On
- Turn Script Logging Off

To debug a script:

1. From the main screen of the Script Editor, click Add As Text to open the Text Editor.
2. Build your script. When your script is complete, click Build to verify the script text.
   The Text Editor displays build information in the bottom section of the window.
3. If the build information includes an error message, click the message to display the Script Editor Error Help dialog box.

   The Script Editor Error Help dialog box displays the error number, a description of the error, and explains how to fix the error.

To edit a script with the Text Editor:

1. Launch the Script Editor.
2. Select the script you want to edit from the script list and click Edit As Text.
   The Text Editor appears.
3. Edit the script as desired, then click Build to verify the script syntax.
4. When you have finished editing the script, click Save or Save As to save a copy of the script text.
5. Click Done to return to the Script Editor, or click Open to edit another script.
**Turn Script Logging On**

Include a `Logging_On` action in your script code to generate a log file. Place the `Logging_On` action at the point you want to begin logging.

**To enter the `Logging_On` action:**

1. From the `Actions` tab, click `Insert`.
2. From the `Action` drop-down menu, select `Logging_On`.
3. Click the `File Path` tab.
4. In the `Constant String` text box, enter the location where you want to store the log file.
5. Click the `Overwrite Previous` tab.
6. Set the Override Previous option.

   When you set the `Override Previous` option to FALSE, the latest log file will not replace the existing file. Instead, a separate log file is created for each log.

   When you set the `Override Previous` option to TRUE, the most recent log file will replace the existing log file.

**NOTE:** Because logging slows the performance of Terminal Emulation and takes up space on your devices, you may not want to include it in end-user scripts. Set the `Override Previous` value to TRUE to keep the log files from getting too large.

7. Click `OK`.

   The code is added to the `Actions` tab.

**Turn Script Logging Off**

If the script exits, logging automatically terminates, so you usually will not need the `Logging_Off` action. However, if you only want to log a portion of the script, you can enter a `Logging_Off` action to stop the logging.

**To enter a `Logging_Off` action:**

1. From the `Actions` tab, click the `Insert` button.
2. From the `Action` drop-down menu, select `Logging_Off`.
3. Click `OK`.

   The code is added to the `Actions` tab.
Chapter 4: Importing and Exporting Scripts

After you finish building a script, your script is automatically saved in the Script Editor. You can also export a script and save it in a specific location on the network.

Scripts are saved as .wls files if you click Export, or as .wlt files if you save them as a text file. Scripts saved in .wls format can't be viewed outside the Script Editor and must be imported back into the Script Editor to view or edit.

This section provides information on importing a .wls file. For information on importing a text file, see Importing a Text File on page 34.

To export a script:
1. From the script list, select the script you want to export.
2. Click Export.
   - Or -
   To save the script as text, select Save As Text.

The Create the Script File dialog box opens.
3 Navigate to the location where you want to export your script.

4 Click **Save**.

To import a script:

1 From the Script Editor, click **Import**.

   The *Select the Script File* dialog box opens.

2 Navigate to and select the script file.

3 Click **Open**.

   The name of the file is imported into the Script Editor.

   Once you have imported the file, you can edit the script.
Chapter 5: Organizing Scripts

The TE Script Editor allows you to organize scripts into groups to make them easier to manage. Groups can be nested inside other groups, and a group can contain both scripts and other groups. To see the scripts in alphabetical order without any script grouping, click the Show as list option in the top left corner of the Script Editor.

Organizing scripts into groups does not change how the scripts are deployed to a device.

**NOTE:** To create groups or move and copy scripts, you should not have the Show as list option enabled.

**To create a group:**
1. From the Script Editor, select the place in the tree where you want to add a group. Click Add Group.

   The Add Group dialog box appears.

2. Type the **New group name** in the text box. If you want the group to be a top-level group instead of nested inside another group, enable the **Top-level group** option. Click OK.

3. The group is added to the tree and you can move scripts into it, rename it, or move it.

**To move a script:**
1. From the Script Editor, select the script in the tree you want to move. Click Move Script.

   The Move Script dialog box appears.

2. Select the group you want to move the script to and click OK.

3. The script is moved to the new group.

**To copy a script:**
1. From the Script Editor, select the script in the tree you want to copy. Click Copy Script.

   The Copy Script dialog box appears.

2. Type a name for the new copy of the script in the text box and click OK.

3. The copy of the script, with a new name, appears in the same group as the original. Move the copy to the desired location.
Chapter 6: Executing Scripts

When a script is created, it has an activation method assigned that specifies how it is activated. Once a script is deployed to the device (using Avalanche, ActiveSync, or a third-party utility), you can create a connection session and run the script.

This section provides information about activating scripts using each of the available activation methods:

- **Select From Menu**
- **On Key Combination**
- **When Session Connects**
- **On Barcode, MSR, or RFID Scan**
- **On Screen Update**
- **From Web Pages**

For information on assigning an activation method to a script, refer to Selecting Activation Methods on page 5.

Select From Menu

With this activation method, you can activate the script from the **Options** or **Term** menu of the TE Client.

**To activate:**

1. Launch the TE Client on the mobile device.
2. From the **Options** or **Term** menu, select **Scripting > Execute Script**.
3 If more than one script is available for the current host profile, select which script you want to use from the list.

NOTE: This option is not available if a script is running for the current session or if the session is not connected.

On Key Combination

With this activation method, the script activates when you press the specified key combination (as long as it is currently possible for script to run).

To activate:
1 Launch the TE Client.

2 Enter the key combination you assigned to execute the script.

When Session Connects

With this activation method, the script activates when a session connects using the specified host profile.

To execute when the session connects:
1 Launch the TE Client.

2 From the Options or Term menu, select Connect.

3 Select the host to which you want to connect.
4  Click OK.

The script runs upon connection.

**On Barcode, MSR, or RFID Scan**

With this activation method, the script activates with each barcode, MSR, or RFID scan.

**On Screen Update**

With this activation method, the script activates (if activation is allowed) every time the text on the emulation screen changes. This includes updates from the host or when the user presses a key and the key value appears on the screen.

**From Web Pages**

You can execute Wavelink scripts from web pages using the `wls` type, followed by the script name. If you plan to launch a script from a web page, do not select a script activation method when you create the script.

**Executing Scripts from Web Pages Example 1**

This example launches a script called `WebAuto` when the web page first loads.

```html
<title>TE70 Test1 - Launch Telnet Scripts</title>
<meta http-equiv="OnStartup" content="wls:WebAuto">
```

**Executing Scripts from Web Pages Example 2**

This example launches a script called `WebClick` when a user clicks the hyperlink "here" on the web page.

```html
<p>Click <a href="wls:WebClick">here</a> to launch the "WebClick" script.</p>
```
Chapter 7: Overview of Actions

This section provides an overview of the actions in the Terminal Emulation Script Editor. The actions are grouped by function. For details on each action, click the name of the action in the tables below.

- Ask User for Integer
- Bitwise Arguments
- Blank Line and Comment Actions
- Boolean Assignments
- Boolean Comparisons
- Call Other Macros
- Conditionals
- Convert Strings to Integers
- ESC Sequence Support
- Field Identifiers and Data
- General Queries
- Get System Information
- Goto Support Actions
- Integer Assignments
- Integer Comparison
- Keyboard
- KeyPress Capture
- Logging
- Macro Existing
- Message
- Number to Character Conversion
- Number/Character Conversion
- Printer Commands
- Reboot
- Scanner Information
- Screen Buttons
- Search the Screen
- Send Characters
- Sounds
- Speech Commands
- Stop Commands
- String Comparisons
- String Handling
- String Variable Assignments
- Suspend
- Waiting
- Web Elements
- WEB Emulation Commands

### Blank Line and Comment Actions

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank_Line</td>
<td>Proceeds to the next instruction without taking any action.</td>
</tr>
<tr>
<td>Comment</td>
<td>Proceeds to the next instruction without taking any action.</td>
</tr>
</tbody>
</table>
Goto Support Actions

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goto</td>
<td>Jumps to the supplied label.</td>
</tr>
<tr>
<td>Label</td>
<td>Label to which a Goto can jump.</td>
</tr>
</tbody>
</table>

Macro Existing

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Exits the script normally.</td>
</tr>
<tr>
<td>Abort</td>
<td>Exits the script immediately.</td>
</tr>
<tr>
<td>Abort_All</td>
<td>Exits all scripts for the session.</td>
</tr>
<tr>
<td>Disconnect</td>
<td>Exits all scripts for the session and disconnects the session.</td>
</tr>
<tr>
<td>Exit_Application</td>
<td>Shuts down the Terminal Emulation application.</td>
</tr>
</tbody>
</table>

Conditionals

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>If</td>
<td>Determines which actions to execute.</td>
</tr>
<tr>
<td>If_Not</td>
<td>Determines which actions to stop executing.</td>
</tr>
<tr>
<td>Else</td>
<td>Starts statements to be executed if an If test fails.</td>
</tr>
<tr>
<td>End If</td>
<td>End of statements to be executed for an If test.</td>
</tr>
<tr>
<td>While</td>
<td>Determines which statements to execute.</td>
</tr>
<tr>
<td>While_Not</td>
<td>Determines which statements to stop executing.</td>
</tr>
<tr>
<td>End While</td>
<td>End of statements to be executed for a While test.</td>
</tr>
<tr>
<td>Continue</td>
<td>Jumps back to the last While statement and retests the test value.</td>
</tr>
</tbody>
</table>
### Chapter 7: Overview of Actions

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Break</td>
<td>Jumps to the first statement following the next <code>EndWhile</code> statement (exiting the loop). In a <code>Case</code> statement, jumps to the first statement following the <code>End_Switch</code> statement.</td>
</tr>
<tr>
<td>Switch</td>
<td>The Switch action block consists of a series of Case actions and an optional Default action, ending with an End_Switch action.</td>
</tr>
<tr>
<td>Case</td>
<td>If the Test parameter matches the value of the previous Switch parameter, the script continues executing until the next Break or End_Switch statement.</td>
</tr>
<tr>
<td>Default</td>
<td>If none of the Case parameter values match the value of the previous Switch parameter, the script continues executing until the next Break or End_Switch statement.</td>
</tr>
<tr>
<td>End_Switch</td>
<td>End of statements to be executed for a Switch test.</td>
</tr>
</tbody>
</table>

### General Queries

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ask_OK</td>
<td>Displays a message in a dialog box with an OK button.</td>
</tr>
<tr>
<td>Ask_OK_Cancel</td>
<td>Displays a message and waits until the user selects a button.</td>
</tr>
<tr>
<td>Ask_Yes_No</td>
<td>Displays a message and waits until the user selects a button.</td>
</tr>
<tr>
<td>Ask_Yes_No_Cancel</td>
<td>Displays a message in a dialog box with a Yes, No, and Cancel button and waits until the user selects a button.</td>
</tr>
<tr>
<td>Run_Application</td>
<td>Starts an application with the flags (optional).</td>
</tr>
</tbody>
</table>
## Send Characters

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keypress_String</td>
<td>Creates one or more key presses to send the supplied string to the Terminal Emulation session.</td>
</tr>
<tr>
<td>Keypress_Key</td>
<td>Sends a single keypress to the Terminal Emulation session.</td>
</tr>
<tr>
<td>Scan_String</td>
<td>Treats the string as scanned data of the type specified.</td>
</tr>
<tr>
<td>Set_Cursor_Position</td>
<td>Moves the cursor to the specified row and column.</td>
</tr>
</tbody>
</table>

## Message

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message</td>
<td>Displays a message on the Terminal Emulation screen.</td>
</tr>
<tr>
<td>Message_Clear</td>
<td>Clears the message on the Terminal Emulation screen.</td>
</tr>
</tbody>
</table>

## Sounds

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beep</td>
<td>Causes the device to beep.</td>
</tr>
<tr>
<td>Play_Sound</td>
<td>Causes the device to play the sound specified by the sound name.</td>
</tr>
</tbody>
</table>

## Waiting

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait_For_Screen_Update</td>
<td>Suspends the current script until the screen has been updated.</td>
</tr>
<tr>
<td>Delay</td>
<td>Suspends the current script until the specified time has passed.</td>
</tr>
</tbody>
</table>
# Chapter 7: Overview of Actions

## Logging

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logging_On</td>
<td>Creates a log file that records all subsequent script execution activity.</td>
</tr>
<tr>
<td>Logging_Off</td>
<td>Turns off logging for the script.</td>
</tr>
</tbody>
</table>

## Call Other Macros

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call</td>
<td>Suspends the current script and executes another script.</td>
</tr>
</tbody>
</table>

## Screen Buttons

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Button_Bitmap_Create_Emulation</td>
<td>Creates a button with the specified bitmap name using the specified text position.</td>
</tr>
<tr>
<td>Button_Bitmap_Create_View</td>
<td>Creates a button with the specified bitmap name using the specified screen position.</td>
</tr>
<tr>
<td>Button_Create_Emulation</td>
<td>Creates a button with the specified text using the specified text position.</td>
</tr>
<tr>
<td>Button_Create_View</td>
<td>Creates a button with the specified text using the specified screen position.</td>
</tr>
<tr>
<td>Button_Remove</td>
<td>Removes buttons created with the previous actions.</td>
</tr>
<tr>
<td>Button_Remove_All</td>
<td>Removes all buttons created with the previous actions.</td>
</tr>
</tbody>
</table>

## Reboot

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reboot</td>
<td>Reboots the device.</td>
</tr>
</tbody>
</table>
### KeyPress Capture

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keypress_Capture</td>
<td>Begins a specific key capture and modifier combination.</td>
</tr>
<tr>
<td>Keypress_Capture_Stop</td>
<td>Stops the specified key capture and modifier combination.</td>
</tr>
<tr>
<td>Keypress_Capture_Stop_All</td>
<td>Stops all key press captures and modifier combinations</td>
</tr>
</tbody>
</table>

### Keyboard

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard_Disable</td>
<td>Disables all keyboards.</td>
</tr>
</tbody>
</table>

### Boolean Assignments

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview of Actions</td>
<td>Set the value of a Boolean variable.</td>
</tr>
<tr>
<td>Boolean_Not</td>
<td>Set the value of a Boolean variable to FALSE if the parameter is TRUE. Set the value to TRUE if the parameter is FALSE.</td>
</tr>
<tr>
<td>Boolean_And</td>
<td>Test each of the parameters and return TRUE if all are TRUE or FALSE if one or more are FALSE.</td>
</tr>
<tr>
<td>Boolean_Or</td>
<td>Test each of the parameters and return TRUE if one or more are true or FALSE if all are FALSE.</td>
</tr>
</tbody>
</table>

### Boolean Comparisons

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean_Equal</td>
<td>Compare the two parameters and return TRUE if they are both TRUE or if they are both FALSE.</td>
</tr>
<tr>
<td>Boolean_Not_Equal</td>
<td>Compare the two parameters and return TRUE if they do not have the same value.</td>
</tr>
</tbody>
</table>
## String Comparisons

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String_Empty</td>
<td>Check the length of the string to determine if it’s an empty string.</td>
</tr>
<tr>
<td>String_Less_Than</td>
<td>Compare the two strings and determine their alphabetical order.</td>
</tr>
<tr>
<td>String_Less_Than_Or_Equal</td>
<td>Compare the two strings and determine whether one precedes the other in alphabetical order, or if they are the same string.</td>
</tr>
<tr>
<td>String_Equal</td>
<td>Compare the two strings and determine if they are both TRUE or are both FALSE.</td>
</tr>
<tr>
<td>String_Greater_Than_Or_Equal</td>
<td>Compare the two strings and determine whether one follows the other in alphabetical order, or they are the same string.</td>
</tr>
<tr>
<td>String_Greater_Than</td>
<td>Compare the two strings and determine whether one follows the other in alphabetical order.</td>
</tr>
<tr>
<td>String_Not_Equal</td>
<td>Compare the two strings and return TRUE if they do not have the same value.</td>
</tr>
</tbody>
</table>

## Field Identifiers and Data

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set_Field_Data_ID</td>
<td>Sets the Data ID for a field.</td>
</tr>
<tr>
<td>Set_Field_Symbology_ID</td>
<td>Sets the Symbology ID for a field.</td>
</tr>
<tr>
<td>Get_Field_Symbology_Operator</td>
<td>Query whether the field data matches the Data ID and/or Symbology ID.</td>
</tr>
<tr>
<td>Set_Field_Append_Scan_Data</td>
<td>Controls whether to append scan data in the field.</td>
</tr>
<tr>
<td>Set_Field_Com_Data_Field</td>
<td>Sets a field to be the Com Data Field for the screen.</td>
</tr>
<tr>
<td>Set_Field_Prefix_Scan_Data</td>
<td>Sets the data prefixed to a field when the field is scanned.</td>
</tr>
<tr>
<td>Get_Field_Append_Scan_Data</td>
<td>Query whether data is appended when the field is scanned.</td>
</tr>
</tbody>
</table>
### Chapter 7: Overview of Actions

#### Action Description

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get_Field_Index</td>
<td>Get the index of a field at the specified row and column.</td>
</tr>
<tr>
<td>Get_Num_Fields</td>
<td>Get the number of fields on the screen.</td>
</tr>
<tr>
<td>Get_Field_Index_Row_Text</td>
<td>Get the index of a field that is in the same row as the text.</td>
</tr>
<tr>
<td>Get_Field_Index_Column_Text</td>
<td>Get the index of a field that is in the same column as the text.</td>
</tr>
<tr>
<td>Get_Field_Row</td>
<td>Get the row number of the field.</td>
</tr>
<tr>
<td>Get_Field_Column</td>
<td>Get the column number of the field.</td>
</tr>
<tr>
<td>Get_Field_Length</td>
<td>Get the length of the field.</td>
</tr>
<tr>
<td>Get_Num_Field_Data_IDs</td>
<td>Get the number of data IDs in a field.</td>
</tr>
<tr>
<td>Get_Num_Field_Symbology_IDs</td>
<td>Get the number of symbology IDs in a field.</td>
</tr>
<tr>
<td>Get_Field_Com_Data_Field</td>
<td>Get the index of the field that is the Com Data Field.</td>
</tr>
<tr>
<td>Get_Field_Data_ID</td>
<td>Gets the Data ID for the specified field.</td>
</tr>
<tr>
<td>Get_Field_Prefix_Scan_Data</td>
<td>Gets the prefixed data for the specified field.</td>
</tr>
</tbody>
</table>

#### Integer Comparison

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number_Less Than</td>
<td>Compares two numbers and determines if one is less than the other.</td>
</tr>
<tr>
<td>Number_LessThan Or Equal</td>
<td>Compares two numbers and determines if one is less than the other or if they are the same.</td>
</tr>
<tr>
<td>Number_Equal</td>
<td>Compare two numbers and determines whether they are equal.</td>
</tr>
<tr>
<td>Number_GreaterThan Or Equal</td>
<td>Compare two numbers and determines if one is greater than the other or if they are equal.</td>
</tr>
</tbody>
</table>
### Chapter 7: Overview of Actions

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number_Greater Than</strong></td>
<td>Compare two numbers and determines if one is greater than the other.</td>
</tr>
<tr>
<td><strong>Number_Not_Equal</strong></td>
<td>Compares two numbers and determines if they are equal.</td>
</tr>
</tbody>
</table>

### Suspend

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Suspend</strong></td>
<td>Suspends the device.</td>
</tr>
<tr>
<td><strong>Wait_For_Screen_Update_With_Timeout</strong></td>
<td>Suspends the current script until the screen has been updated.</td>
</tr>
</tbody>
</table>

### Search the Screen

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Search_Screen</strong></td>
<td>Searches the screen for the supplied text.</td>
</tr>
</tbody>
</table>

### WEB Emulation Commands

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Web_Navigate</strong></td>
<td>Navigates WEB emulation to the URL provided.</td>
</tr>
<tr>
<td><strong>Web_Navigate_Frame</strong></td>
<td>Navigates WEB emulation to the URL provided within the indicated frame.</td>
</tr>
<tr>
<td><strong>Web_Navigate_Post_Data</strong></td>
<td>Navigates WEB emulation to the URL provided.</td>
</tr>
<tr>
<td><strong>Web_Scripting</strong></td>
<td>Instructs WEB emulation to execute the scripting information.</td>
</tr>
<tr>
<td><strong>Web_Search_Source</strong></td>
<td>Searches the page source of the current WEB emulation page.</td>
</tr>
</tbody>
</table>
### Speech Commands

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speech_From_Text_Available</td>
<td>Determines whether text-to-speech is supported.</td>
</tr>
<tr>
<td>Speech_From_Text</td>
<td>Converts text into sound and plays it on the computer.</td>
</tr>
<tr>
<td>Speech_To_Text_Available</td>
<td>Determines whether speech-to-text is supported.</td>
</tr>
<tr>
<td>Speech_To_Text</td>
<td>Listens to the user speak and returns the text equivalent of what he/she said in the string variable.</td>
</tr>
<tr>
<td>Speech_To_Text_No_Wait</td>
<td>Listens to the user speak and returns the text equivalent.</td>
</tr>
<tr>
<td>Speech_To_Text_Cancel</td>
<td>Provides a way for the script to perform other functions while the speech-to-text action occurs.</td>
</tr>
<tr>
<td>Speech_Setting_Available</td>
<td>Identifies speech settings by case-insensitive name strings.</td>
</tr>
<tr>
<td>Speech_Change_Setting</td>
<td>Changes the speech setting to the specified value.</td>
</tr>
<tr>
<td>Speech_Get_Setting</td>
<td>Gets the value of the speech setting.</td>
</tr>
<tr>
<td>Speech_Get_Setting_Max</td>
<td>Gets the largest value for the speech setting.</td>
</tr>
<tr>
<td>Speech_Find_Setting_Value</td>
<td>Searches all possible value descriptions for the speech setting.</td>
</tr>
<tr>
<td>Speech_Get_Setting_Value_Desc</td>
<td>Gets a description of the speech setting value.</td>
</tr>
<tr>
<td>Speech_To_Text_Get_User_Name</td>
<td>Gets the user name.</td>
</tr>
<tr>
<td>Speech_To_Text_Change_User_Name</td>
<td>Changes the user name being used by the speech-to-text engine.</td>
</tr>
</tbody>
</table>
### Chapter 7: Overview of Actions

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Speech_From_Text_Error_Desc</strong></td>
<td>Gets an error description for the last speech-from-text action.</td>
</tr>
<tr>
<td><strong>Speech_To_Text_Error_Desc</strong></td>
<td>Gets an error description for the last speech-to-text action.</td>
</tr>
<tr>
<td><strong>Speech_From_Text_Cancel</strong></td>
<td>Provides a way for the script to perform other functions while the text-to-speech action occurs.</td>
</tr>
<tr>
<td><strong>Speech_Get_Confidence_Level</strong></td>
<td>Gets the confidence value for the last Speech_To_Text action.</td>
</tr>
</tbody>
</table>

### Stop Commands

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Keypress_Capture_Stop</strong></td>
<td>Stops capturing the specified key and modifier combination.</td>
</tr>
<tr>
<td><strong>Cancel_Other_Scripts</strong></td>
<td>Cancels all other scripts for the session with the script name.</td>
</tr>
</tbody>
</table>

### Printer Commands

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Printer_Data</strong></td>
<td>Sends data directly to the currently defined printer.</td>
</tr>
<tr>
<td><strong>Printer_Repeat</strong></td>
<td>Instructs the printer to reprint the last item printed.</td>
</tr>
<tr>
<td><strong>Printer_Cancel</strong></td>
<td>Instructs the printer to discard all Printer_Data data already received.</td>
</tr>
</tbody>
</table>
# Get System Information

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview of Actions</td>
<td>Gets the number of columns on the screen.</td>
</tr>
<tr>
<td>Get_Screen_Rows</td>
<td>Gets the number of rows on the screen.</td>
</tr>
<tr>
<td>Get_Position_Column</td>
<td>Gets the column number on which the cursor is currently located.</td>
</tr>
<tr>
<td>Get_Position_Row</td>
<td>Get the row number on which the cursor is currently located.</td>
</tr>
<tr>
<td>Get_Session_Number</td>
<td>Get the number for the session in which this script is executing</td>
</tr>
<tr>
<td>Get_Time</td>
<td>Get the amount of time passed since January 1, 2000.</td>
</tr>
<tr>
<td>Get_Time_Since_Reset</td>
<td>Gets the amount of time since the last reboot.</td>
</tr>
<tr>
<td>Overview of Actions</td>
<td>Gets the MAC address of the device.</td>
</tr>
<tr>
<td>Get_IP_Address</td>
<td>Gets the IP address of the device.</td>
</tr>
<tr>
<td>Get_Field_Symbology_ID</td>
<td>Gets the symbology ID of the specified field.</td>
</tr>
<tr>
<td>Get_Screen_Text</td>
<td>Gets the text at the specified location.</td>
</tr>
<tr>
<td>Get_Screen_Text_Length</td>
<td>Gets the specified amount of text on the screen.</td>
</tr>
<tr>
<td>Get_Screen_Text_Columns</td>
<td>Get text from a row on the screen, starting at a specific column.</td>
</tr>
<tr>
<td>Get_Workstation_ID</td>
<td>Gets the Workstation ID of the device.</td>
</tr>
<tr>
<td>Get_Avalanche_Property_Value</td>
<td>Gets the value of the specified Avalanche property.</td>
</tr>
</tbody>
</table>

# Scanner Information

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get_Scan_Type_Value</td>
<td>Get the number value of the supplied scan type name.</td>
</tr>
<tr>
<td>Get_Scan_Type_Name</td>
<td>Gets the name of the scan type.</td>
</tr>
</tbody>
</table>
### String Handling

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String_Length</td>
<td>Get the number of characters in a string.</td>
</tr>
<tr>
<td>String_Find_First</td>
<td>Finds the first instance of the substring inside the string, and returns the position where that substring starts.</td>
</tr>
<tr>
<td>String_Find_Last</td>
<td>Finds the last instance of the substring inside the string, and returns the position where that substring starts.</td>
</tr>
</tbody>
</table>

### Integer Assignments

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number_Set</td>
<td>Set the value of a number variable.</td>
</tr>
<tr>
<td>Number_Plus</td>
<td>Add two numbers together and return the sum.</td>
</tr>
<tr>
<td>Number_Minus</td>
<td>Subtract the second term from the first term to get the difference.</td>
</tr>
<tr>
<td>Number_Multiply</td>
<td>Multiply the first term by the second term and returns the product.</td>
</tr>
<tr>
<td>Number_Divide</td>
<td>Divide the first term by the second term and return the product.</td>
</tr>
<tr>
<td>Number_Divide_Remainder</td>
<td>Divide the first term by the second term and return the remainder. For example, 7 divided by 3 would return a remainder of 1.</td>
</tr>
</tbody>
</table>

### Convert Strings to Integers

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String_To_Number_Binary</td>
<td>Get a string’s binary representation.</td>
</tr>
<tr>
<td>String_To_Number_Octal</td>
<td>Gets a string’s octal representation.</td>
</tr>
<tr>
<td>String_To_Number_Decimal</td>
<td>Gets a string’s decimal representation.</td>
</tr>
<tr>
<td>String_To_Number_Hexadecimal</td>
<td>Gets a string’s hexadecimal representation.</td>
</tr>
</tbody>
</table>
Chapter 7: Overview of Actions

Ask User for Integer

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ask_Number</td>
<td>Displays a dialog box asking the user for a decimal number.</td>
</tr>
</tbody>
</table>

Number/Character Conversion

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character_To_Number</td>
<td>Converts the character at position Index in the string into the number value for that character.</td>
</tr>
</tbody>
</table>

Bitwise Arguments

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bitwise_And</td>
<td>The resulting number will have a bit set when both input numbers have that bit set.</td>
</tr>
<tr>
<td>Bitwise_Or</td>
<td>The resulting number will have a bit set when either input numbers has that bit set (inclusive or).</td>
</tr>
<tr>
<td>Bitwise_Xor</td>
<td>The resulting number will have a bit set when exactly one input number has that bit set (exclusive or).</td>
</tr>
<tr>
<td>Bitwise_Not</td>
<td>The resulting number will have a bit set when the input number does not have that bit set (ones complement).</td>
</tr>
</tbody>
</table>

Web Elements

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web_Get_Source</td>
<td>Returns the HTML code of the search string.</td>
</tr>
<tr>
<td>Web_Get_Current_Element</td>
<td>Returns the HTML code for the Web element with the focus.</td>
</tr>
</tbody>
</table>

ESC Sequence Support

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Escape_Sequence</td>
<td>Handles the supplied Wavelink Custom or Telxon ESC Sequence for all emulation types.</td>
</tr>
</tbody>
</table>
## String Variable Assignments

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String_Set</td>
<td>Gets the specified string.</td>
</tr>
<tr>
<td>String.Combine</td>
<td>Returns the concatenated value of two strings.</td>
</tr>
<tr>
<td>String.Left</td>
<td>Returns the specified characters of the input string.</td>
</tr>
<tr>
<td>String.Right</td>
<td>Returns the specified characters of the input string.</td>
</tr>
<tr>
<td>String.Middle</td>
<td>Returns the specified characters of the input string.</td>
</tr>
<tr>
<td>String.Upper</td>
<td>Converts the specified text to uppercase letters.</td>
</tr>
<tr>
<td>String.Lower</td>
<td>Converts the specified text to lowercase letters.</td>
</tr>
<tr>
<td>String.Replace</td>
<td>Replaces the specified text with another string.</td>
</tr>
<tr>
<td>String.Only_Characters</td>
<td>Gets a string with the specified characters.</td>
</tr>
<tr>
<td>String.Strip_Characters</td>
<td>Strips the specified characters from the string.</td>
</tr>
<tr>
<td>String.Trim_Spaces_Start</td>
<td>Gets the specified text with all tabs and spaces deleted.</td>
</tr>
<tr>
<td>String.Trim_Spaces_End</td>
<td>Gets the specified text with all tabs and spaces deleted.</td>
</tr>
<tr>
<td>Number_To_String_Binary</td>
<td>Gets the binary representation of the specified number.</td>
</tr>
<tr>
<td>Number_To_String_Octal</td>
<td>Gets the octal representation of the specified number.</td>
</tr>
<tr>
<td>Number_To_String.Decimal</td>
<td>Gets the decimal representation of the specified number.</td>
</tr>
<tr>
<td>Number_To_String_Hexadecimal_Lowercase</td>
<td>Gets the hexadecimal representation of the specified number.</td>
</tr>
<tr>
<td>Number_To_String_Hexadecimal_Uppercase</td>
<td>Gets the hexadecimal representation of the specified number.</td>
</tr>
</tbody>
</table>
### Action Description

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ask_String</td>
<td>Displays a dialog box asking the user for a string.</td>
</tr>
<tr>
<td>Ask_String_Password</td>
<td>Displays a dialog box asking the user for a string.</td>
</tr>
<tr>
<td>Ask_String_Uppercase</td>
<td>Displays a dialog box asking the user for a string.</td>
</tr>
<tr>
<td>Ask_String_Lowercase</td>
<td>Displays a dialog box asking the user for a string.</td>
</tr>
</tbody>
</table>

### Number to Character Conversion

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number_To_Character</td>
<td>Converts the specified number to the character value.</td>
</tr>
</tbody>
</table>
Abort

Exits the script immediately. If this script was started by another script, the calling script’s variables are not updated and the calling script resumes.

Example

Script(Abort_Test)
Activate(From_Menu)
Comment: This script doesn’t do anything.
Abort

See Also

Return, Abort_All, Disconnect, Exit_Application
Chapter 7: Overview of Actions

Abort_All

Exits all scripts for the session.

Example

Script(Abort_All_Test)
Activate(From_Menu)
  Comment: This script causes all of the session’s scripts to abort.
  Abort_All

See Also

Return, Abort, Disconnect, Exit_Application, Reboot
Ask_Number

Displays a dialog box asking the user for a decimal number. The supplied default value is returned if the user cancels the dialog.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Text</td>
<td>The message displayed in the box.</td>
</tr>
<tr>
<td>Title Text</td>
<td>The title of the message box.</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>The smallest value of the number.</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>The largest value of the number.</td>
</tr>
<tr>
<td>Default Value</td>
<td>The initial value in the message box.</td>
</tr>
</tbody>
</table>

Format

`Ask_Number ("Message Text", "Title Text", Minimum Value, Maximum Value, Default Value)`

Return Value

Returns the number supplied by the user.

Example

```wolfram
Script(Number_Convert)
String(strEntered)
String(strBinary)
String(strHexLower)
String(strHexUpper)
String(strOctal)
Number(numEntered)
Activate(From_Menu)
    numEntered = Ask_Number("Enter the decimal number to convert", "Number_Convert", -22, 2000000000, 31)
    strEntered = Number_To_String_Decimal(numEntered)
    strBinary = Number_To_String_Binary(numEntered)
    strHexLower = Number_To_String_Hexadecimal_Lowercase(numEntered)
    strHexUpper = Number_To_String_Hexadecimal_Uppercase(numEntered)
    strOctal = Number_To_String_Octal(numEntered)
    Ask_OK(strBinary, String_Combine("Binary value of", strEntered))
    Ask_OK(strHexLower, String_Combine("Hex (lower case) value of ", strEntered))
```
Ask_OK(strHexUpper, String_Combine("Hex (upper case) value of ", strEntered))
Ask_OK(strOctal, String_Combine("Octal value of ", strEntered))
Return

See Also

Ask_OK, Ask_OK_Cancel, Ask_Yes_No, Ask_String, Ask_String_Password, Ask_String_Uppercase, Ask_String_Lowercase, Ask_Yes_No_Cancel, Message
Chapter 7: Overview of Actions

Ask_OK

Displays a message in a box with an OK button and waits until the user presses the button.

Parameters

- **Message Text**: The message displayed in the box.
- **Title Text**: The title of the message box.

Format

```
Ask_Ok(Message Text, "Title Text")
```

Example

```
Script( Test_Ask_Ok )
Activate( From_Menu )
Message_Clear
Ask_OK( "Press OK and the script will end.", "Test_Ask_Ok" )
Return
```

See Also

- Ask_OK_Cancel, Ask_Yes_No, Ask_Yes_No_Cancel, Message
Ask_OK_Cancel

Displays a message in a dialog box with an OK and Cancel button and waits until the user selects a button.

Parameters

- Message Text: The message that the script displays in the message box.
- Title Text: The message that the script displays in the title bar of the message box.
- Make Cancel Default: Indicates whether the Cancel button is the default button.

Format

Ask_OK_Cancel ("Message Text", "Title Text", Make Cancel Default)

Return Value

Returns a Boolean. TRUE if the user selects OK, returns FALSE if the user selects Cancel.

Example

Script(Ask_OK_Cancel_Test)
Boolean(bResult)
Activate(From_Menu)
   bResult = Ask_OK_Cancel("OK for TRUE, Cancel for FALSE", "Ask_OK_Cancel_Test", FALSE)
   If(bResult)
      Message("Selected: OK", 5)
   Else
      Message("Selected: Cancel", 5)
   End_If
Return

See Also

Ask_Yes_No, Ask_Yes_No_Cancel, Ask_OK, Ask_String, Ask_String_Password, Ask_String_Uppercase, Ask_String_Lowercase, Message, Ask_Number
Ask_String

Displays a dialog box asking the user for a string. The supplied default string is returned (unaltered) if the user cancels the dialog.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Text</td>
<td>The message displayed in the dialog box.</td>
</tr>
<tr>
<td>Title Text</td>
<td>The title displayed in the dialog box.</td>
</tr>
<tr>
<td>Minimum Length of String</td>
<td>The minimum number of characters the string must have.</td>
</tr>
<tr>
<td>Maximum Length of String</td>
<td>The maximum number of characters the string can have.</td>
</tr>
<tr>
<td>Default String</td>
<td>The initial value in the message box which can be changed by the user.</td>
</tr>
</tbody>
</table>

Format

Ask_String ("Enter a string", "Ask_String_Test", Minimum Length of String, Maximum Length of String, Default String)

Return Value

Returns the string supplied by the user.

Example

```plaintext
Script(AssK_String_Test)  
String(strEntered)  
Activate(From_Menu)  
    strEntered = Ask_String("Enter a string", "Ask_String_Test", 1, 99, ")  
    Ask_OK(strEntered, "You Entered")  
Return
```

See Also

String_Set, String_Combine, String_Left, String_Right, String_Middle, String_Upper, String_Lower, String_Replace, String_Only_Characters, String_Strip_Characters, String_Trim_Spaces_Start, String_Trim_Spaces_End, Number_To_String_Binary, Number_To_String_Octal, Number_To_String.Decimal, Number_To_String_Hexadecimal_Lowercase, Number_To_String_Hexadecimal_Uppercase, Ask_String_Password, Ask_String_Uppercase, Ask_String_Lowercase, String_Equal
**Ask_String_Lowercase**

Displays a dialog box asking the user for a string. Any uppercase letters are converted to lowercase characters. The supplied default string is returned (unaltered) if the user cancels the dialog.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Message Text</strong></td>
<td>The message displayed in the dialog box.</td>
</tr>
<tr>
<td><strong>Title Text</strong></td>
<td>The title displayed in the dialog box.</td>
</tr>
<tr>
<td><strong>Minimum Length of String</strong></td>
<td>The minimum number of characters the string must have.</td>
</tr>
<tr>
<td><strong>Maximum Length of String</strong></td>
<td>The maximum number of characters the string can have.</td>
</tr>
<tr>
<td><strong>Default String</strong></td>
<td>The initial value in the message box which can be changed by the user.</td>
</tr>
</tbody>
</table>

**Format**

```
Ask_String_Lowercase ("Enter a string", "Ask_String_Test", Minimum Length of String, Maximum Length of String, Default String)
```

**Return Value**

Returns the string supplied by the user.

**Example**

```wolfram
Script(Ask_String_Lowercase_Test)
String(strEntered)
Activate(From_Menu)
    strEntered = Ask_String_Lowercase("The string you enter will be lowercase", "Ask_String_Lowercase", 1, 99, "")
    Ask_OK(strEntered, "The lower case string")
Return
```

**See Also**

String_Set, String_Combine, String_Left, String_Right, String_Middle, String_Upper, String_Lower, String.Replace, String_Only_Characters, String_Strip_Characters, String_Trim_Spaces_Start, String_Trim_Spaces_End, Number_To_String_Binary, Number_To_String_Octal, Number_To_String.Decimal, Number_To_String_Hexadecimal_Lowercase, Number_To_
String_Hexadecimal_Uppercase, Ask_String, Ask_String_Password, Ask_String_Uppercase, String_Equal
Chapter 7: Overview of Actions

Ask_String_Password

Displays a dialog box asking the user for a string. The string is displayed as a password (a series of asterisks). The supplied default string is returned (unaltered) if the user cancels the dialog.

Parameters

- **Message Text**
  - The message displayed in the dialog box.
- **Title Text**
  - The title displayed in the dialog box.
- **Minimum Length of String**
  - The minimum number of characters the string must have.
- **Maximum Length of String**
  - The maximum number of characters the string can have.
- **Default String**
  - The initial value in the message box which can be changed by the user.

Format

```
Ask_String_Password ("Enter a string", "Ask_String_Test", Minimum Length of String, Maximum Length of String, Default String)
```

Return Value

Returns the string supplied by the user.

Example

```
Script(Ask_String_Password_Test)
String(strEntered)
Activate(From_Menu)
    strEntered = Ask_String_Password("Enter a password", "Ask_String_Password_Test", 1, 99, "")
    Ask_OK(strEntered, "The password is")
Return
```

See Also

- String_Set, String_Combine, String_Left, String_Right, String_Middle, String_Upper, String_Lower, String_Replace, String_Only_Characters, String_Strip_Characters, String_Trim_Spaces_Start, String_Trim_Spaces_End, Number_To_String_Binary, Number_To_String_Octal, Number_To_String_Decimal, Number_To_String_Hexadecimal_Lowercase, Number_To_
String_Hexadecimal_Uppercase, Ask_String, Ask_String_Uppercase, Ask_String_Lowercase, String_Equal
Ask_String_Uppercase

Displays a dialog box asking the user for a string. Any lowercase letters are converted to uppercase characters. The supplied default string is returned (unaltered) if the user cancels the dialog.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Text</td>
<td>The message displayed in the dialog box.</td>
</tr>
<tr>
<td>Title Text</td>
<td>The title displayed in the dialog box.</td>
</tr>
<tr>
<td>Minimum Length of String</td>
<td>The minimum number of characters the string must have.</td>
</tr>
<tr>
<td>Maximum Length of String</td>
<td>The maximum number of characters the string can have.</td>
</tr>
<tr>
<td>Default String</td>
<td>The initial value in the message box which can be changed by the user.</td>
</tr>
</tbody>
</table>

Format

Ask_String_Uppercase ("Enter a string", "Ask_String_Test", Minimum Length of String, Maximum Length of String, Default String)

Return Value

Returns the string supplied by the user.

Example

Script(Ask_String_Uppercase_Test)
String(strEntered)
Activate(From_Menu)
    strEntered = Ask_String_Uppercase("The string you enter will be uppercase", "Ask_String_Uppercase", 1, 99, "")
    Ask_OK(strEntered, "The upper case string")
Return

See Also

String_Set, String_Combine, String_Left, String_Right, String_Middle, String_Upper, String_Lower, String_Replace, String_Only_Characters, String_Strip_Characters, String_Trim_Spaces_Start, String_Trim_Spaces_End, Number_To_String_Binary, Number_To_String_Octal, Number_To_String_Decimal, Number_To_String_Hexadecimal_Lowercase, Number_To_
String_Hexadecimal_Uppercase, Ask_String, Ask_String_Password, Ask_String_Lowercase, String_Equal
Chapter 7: Overview of Actions

Ask_Yes_No

Displays a message in a box with a Yes and No button and waits until the user selects a button.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Text</td>
<td>The message that the script displays in the message box.</td>
</tr>
<tr>
<td>Title Text</td>
<td>The message that the script displays in the title bar of the message box.</td>
</tr>
<tr>
<td>Make No Default</td>
<td>Indicates whether No is the default button. If this is FALSE, the Yes button is the default button.</td>
</tr>
</tbody>
</table>

Format

```
Ask_Yes_No ("Message Text", "Title Text", TRUE)
```

Return Value

Returns a Boolean. TRUE if the user selects Yes, returns FALSE if the user selects No.

Example

```
Script(Ask_Yes_No_Test)
Boolean(bResult)
Activate(From_Menu)
    bResult = Ask_Yes_No("Yes for TRUE, No for FALSE", "Ask_Yes_No_Test", FALSE)
    If(bResult)
        Message("Selected: Yes", 5)
    Else
        Message("Selected: No", 5)
    End_If
Return
```

See Also

Ask_OK_Cancel, Ask_Yes_No_Cancel, Ask_OK, Ask_String, Ask_String_Password, Ask_String_Uppercase, Ask_String_Lowercase, Message, Ask_Number
Ask_Yes_No_Cancel

Displays a message in a dialog box with a Yes, No, and Cancel button and waits until the user selects a button.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Text</td>
<td>The message displayed in the message box.</td>
</tr>
<tr>
<td>Title Text</td>
<td>The title displayed on the message box.</td>
</tr>
<tr>
<td>Make No Default</td>
<td>Indicates whether No is the default button. If this is FALSE, the Yes button is the default button.</td>
</tr>
</tbody>
</table>

Format

Ask_Yes_No_Cancel ("Message Text", "Title Text", Make No Default)

Return Value

Returns 2 if the user presses Yes, 1 if the user presses No, and 0 if the user presses Cancel.

Example

```plaintext
Script(Ask_Yes_No_Cancel_Test)
Number(nResult)
Activate(From_Menu)
    nResult = Ask_Yes_No_Cancel("Select Yes, No, or Cancel", "Ask_Yes_No_Cancel", FALSE)
    If(Number_Equal(nResult, 0))
        Message("Cancel", 3)
    Else
        If(Number_Equal(nResult, 1))
            Message("No", 3)
        Else
            Message("Yes", 3)
    End_If
End_If
Return
```

See Also

Ask_Number, Ask_OK, Ask_OK_Cancel, Ask_Yes_No, Ask_String, Ask_String_Password, Ask_String_Uppercase, Ask_String_Lowercase, Message
Chapter 7: Overview of Actions

Beep

Causes the device to beep. A frequency of 1000 is a good default.

Parameters

- **Frequency (Hz)**: The frequency of the beep in cycles per second.
- **Duration (Milliseconds)**: The length of the beep. A value of 1000 would be one second.
- **Volume**: The volume at which the beep is played. The volume is a value between 0 and 9, where 0 is the softest and 9 is the loudest.

Format

Beep (Frequency, Duration, Volume)

Example

Script(Beep_Test)
  Activate(From_Menu)
  Comment: Beep for one-half second at 1000Hz, maximum volume:
  Beep(1000, 500, 9)
  Comment: Beep for three seconds at 2500Hz, medium volume:
  Beep(2500, 3000, 5)
  Return

See Also

Play_Sound, Speech_From_Text
**Bitwise_And**

The resulting number will have a bit set when both input numbers have that bit set.

**Parameters**

- **Value1**  Number; Number 1
- **Value2**  Number; Number 2

**Format**

```plaintext
Bitwise_And (Value1, Value2)
```

**Example**

```plaintext
Script( Bitwise_And_Test )
String( strEntered )
String( strMessage )
String( strBinary1 )
String( strBinary2 )
String( strBinaryResult )
String( strDecimal1 )
String( strDecimal2 )
String( strDecimalResult )
String( strHexUpper1 )
String( strHexUpper2 )
String( strHexUpperResult )
Number( nHexadecimal1 )
Number( nHexadecimal2 )
Number( nHexResult )
Activate( From_Menu )

strEntered = Ask_String( "Enter first hexadecimal number", "Bitwise_And", 1, 8, "A5" )
nHexadecimal1 = String_To_Number_Hexadecimal( strEntered )
strHexUpper1 = strEntered

strEntered = Ask_String( "Enter second hexadecimal number", "Bitwise_And", 1, 8, "83" )
nHexadecimal2 = String_To_Number_Hexadecimal( strEntered )
strHexUpper2 = strEntered

nHexResult = Bitwise_And( nHexadecimal1, nHexadecimal2 )
strHexUpperResult = Number_To_String_Hexadecimal_Uppercase(nHexResult)
```
strBinary1 = Number_To_String_Binary( nHexadecimal1 )
strBinary2 = Number_To_String_Binary( nHexadecimal2 )
strBinaryResult = Number_To_String_Binary( nHexResult )
strDecimal1 = Number_To_String_Decimal( nHexadecimal1 )
strDecimal2 = Number_To_String_Decimal( nHexadecimal2 )
strDecimalResult = Number_To_String_Decimal( nHexResult )

strMessage = String_Combine( strHexUpper1, " AND " )
strMessage = String_Combine( strMessage, strHexUpper2 )
strMessage = String_Combine( strMessage, " = " )
strMessage = String_Combine( strMessage, strHexUpperResult )
Ask_OK( strMessage, "Bitwise And: Hexadecimal" )

strMessage = String_Combine( strBinary1, " AND " )
strMessage = String_Combine( strMessage, strBinary2 )
strMessage = String_Combine( strMessage, " = " )
strMessage = String_Combine( strMessage, strBinaryResult )
Ask_OK( strMessage, "Bitwise And: Binaryl" )

strMessage = String_Combine( strDecimal1, " AND " )
strMessage = String_Combine( strMessage, strDecimal2 )
strMessage = String_Combine( strMessage, " = " )
strMessage = String_Combine( strMessage, strDecimalResult )
Ask_OK( strMessage, "Bitwise And: Decimal" )

Return
Bitwise_Not

The resulting number will have a bit set when the input number does not have that bit set (ones complement).

Parameters

Value1  Number; Number

Format

Bitwise_And (Value1)

Example

Script( Bitwise_Not_Test )
String( strEntered )
String( strMessage )
String( strBinary )
String( strBinaryResult )
String( strDecimal )
String( strDecimalResult )
String( strHexUpper )
String( strHexUpperResult )
Number( nHexadecimal )
Number( nHexResult )
Activate( From_Menu )
    strEntered = Ask_String( "Enter hexadecimal number", "Bitwise_Not", 1, 8, "AF" )
    nHexadecimal = String_To_Number_Hexadecimal( strEntered )
    strHexUpper = strEntered

    nHexResult = Bitwise_Not( nHexadecimal )
    strHexUpperResult = Number_To_String_Hexadecimal_Uppercase (nHexResult)

    strBinary = Number_To_String_Binary( nHexadecimal )
    strBinaryResult = Number_To_String_Binary( nHexResult )
    strDecimal = Number_To_String_Decimal( nHexadecimal )
    strDecimalResult = Number_To_String_Decimal( nHexResult )

    strMessage = String_Combine("Not ", strHexUpper )
    strMessage = String_Combine(strMessage, " = " )
    strMessage = String_Combine(strMessage, strHexUpperResult )
    Ask_OK( strMessage, "Bitwise_Not: Hexadecimal" )
strMessage = String_Combine("Not ", strBinary )
strMessage = String_Combine(strMessage, " = ")
strMessage = String_Combine(strMessage, strBinaryResult )
Ask_OK( strMessage, "Bitwise_Not: Binaryl" )

strMessage = String_Combine("Not ", strDecimal )
strMessage = String_Combine(strMessage, " = ")
strMessage = String_Combine(strMessage, strDecimalResult )
Ask_OK( strMessage, "Bitwise_Not: Decimal" )
Return
Chapter 7: Overview of Actions

Bitwise_Or

The resulting number will have a bit set when either input numbers has that bit set (inclusive or).

Parameters

Value1  Number; Number 1
Value2  Number; Number 2

Format

Bitwise_Or (Value1, Value2)

Example

Script( Bitwise_Or_Test )
String( strEntered )
String( strMessage )
String( strBinary1 )
String( strBinary2 )
String( strBinaryResult )
String( strDecimal1 )
String( strDecimal2 )
String( strDecimalResult )
String( strHexUpper1 )
String( strHexUpper2 )
String( strHexUpperResult )
Number( nHexadecimal1 )
Number( nHexadecimal2 )
Number( nHexResult )
Activate( From_Menu )
    strEntered = Ask_String( "Enter first hexadecimal number", "Bitwise_Or", 1, 8, "A5" )
    nHexadecimal1 = String_To_Number_Hexadecimal( strEntered )
    strHexUpper1 = strEntered
    strEntered = Ask_String( "Enter second hexadecimal number", "Bitwise_Or", 1, 8, "5A" )
    nHexadecimal2 = String_To_Number_Hexadecimal( strEntered )
    strHexUpper2 = strEntered
    nHexResult = Bitwise_Or( nHexadecimal1, nHexadecimal2 )
    strHexUpperResult = Number_To_String_Hexadecimal_Uppercase (nHexResult)
strBinary1 = Number_To_String_Binary( nHexadecimal1 )
strBinary2 = Number_To_String_Binary( nHexadecimal2 )
strBinaryResult = Number_To_String_Binary( nHexResult )
strDecimal1 = Number_To_String_Decimal( nHexadecimal1 )
strDecimal2 = Number_To_String_Decimal( nHexadecimal2 )
strDecimalResult = Number_To_String_Decimal( nHexResult )

strMessage = String_Combine(strHexUpper1, " Or " )
strMessage = String_Combine(strMessage, strHexUpper2 )
strMessage = String_Combine(strMessage, " = " )
Ask_OK( strMessage, "Bitwise Or: Hexadecimal" )

strMessage = String_Combine(strBinary1, " Or " )
strMessage = String_Combine(strMessage, strBinary2 )
strMessage = String_Combine(strMessage, " = " )
Ask_OK( strMessage, "Bitwise Or: Binary" )

strMessage = String_Combine(strDecimal1, " Or " )
strMessage = String_Combine(strMessage, strDecimal2 )
strMessage = String_Combine(strMessage, " = " )
Ask_OK( strMessage, "Bitwise Or: Decimal" )
Return
**Bitwise_Xor**

The resulting number will have a bit set when exactly one input number has that bit set (exclusive or).

**Parameters**

Value1  Number; Number 1  
Value2  Number; Number 2

**Format**

Bitwise_Xor (Value1, Value2)

**Example**

```script
Script( Bitwise_Xor_Test )
String( strEntered )
String( strMessage )
String( strBinary1 )
String( strBinary2 )
String( strBinaryResult )
String( strDecimal1 )
String( strDecimal2 )
String( strDecimalResult )
Number( nHexadecimal1 )
Number( nHexadecimal2 )
Number( nHexResult )
Activate( From_Menu )
  strEntered = Ask_String( "Enter first hexadecimal number", "Bitwise_Xor", 1, 8, "A5" )
  nHexadecimal1 = String_To_Number_Hexadecimal( strEntered )
  strHexUpper1 = strEntered

  strEntered = Ask_String( "Enter second hexadecimal number", "Bitwise_Xor", 1, 8, "FF" )
  nHexadecimal2 = String_To_Number_Hexadecimal( strEntered )
  strHexUpper2 = strEntered

  nHexResult = Bitwise_Xor( nHexadecimal1, nHexadecimal2 )
  strHexUpperResult = Number_To_String_Hexadecimal_Uppercase( nHexResult )
```
strBinary1 = Number_To_String_Binary( nHexadecimal1 )
strBinary2 = Number_To_String_Binary( nHexadecimal2 )
strBinaryResult = Number_To_String_Binary( nHexResult )
strDecimal11 = Number_To_String_Decimal( nHexadecimal1 )
strDecimal2 = Number_To_String_Decimal( nHexadecimal2 )
strDecimalResult = Number_To_String_Decimal( nHexResult )

strMessage = String_Combine( strHexUpper1, " Xor " )
strMessage = String_Combine( strMessage, strHexUpper2 )
strMessage = String_Combine( strMessage, " = " )
Ask_OK( strMessage, "Bitwise_Xor: Hexadecimal" )

strMessage = String_Combine( strBinary1, " Xor " )
strMessage = String_Combine( strMessage, strBinary2 )
strMessage = String_Combine( strMessage, " = " )
Ask_OK( strMessage, "Bitwise_Xor: Binaryl" )

strMessage = String_Combine( strDecimal11, " Xor " )
strMessage = String_Combine( strMessage, strDecimal2 )
strMessage = String_Combine( strMessage, " = " )
Ask_OK( strMessage, "Bitwise_Xor: Decimal" )

Return
Blank_Line

Proceeds to the next instruction without taking any action.

Example

Script(Blank_Line_Test)
Activate(From_Menu)
  Comment: This script has some blank lines:

Return

See Also

Comment
Boolean_And

Test each of the parameters and return TRUE if all parameters are TRUE. Return FALSE if one or more parameters are FALSE. One to five parameters may be used for this action.

All tests will be evaluated each time this action is taken. Use Boolean variables as the parameters instead of actions to make the script easier to read and understand.

Parameters

Test1  A Boolean variable, constant, or action.
Test2  An optional Boolean variable, constant, or action.
Test3  An optional Boolean variable, constant, or action.
Test4  An optional Boolean variable, constant, or action.
Test5  An optional Boolean variable, constant, or action.

Format

Boolean_And (Test1, Test2, Test3, etc.)

Return Value

Returns a Boolean. TRUE if all test values are TRUE; returns FALSE otherwise.

Example

Script(Boolean_And_Test)
Boolean(bResultAll)
Boolean(bResult1)
Boolean(bResult2)
Activate(From_Menu)
  bResult1 = Ask_OK_Cancel("Hit OK in every message box", "Message 1", FALSE)
  bResult2 = Ask_OK_Cancel("Hit OK again", "Message 2", FALSE)
  bResultAll = Boolean_And(bResult1, bResult2)
  If(bResultAll)
    Message("bResultAll is TRUE", 5)
  Else
    Message("bResultAll is FALSE", 5)
  End_If
Return
See Also

Boolean_Set, Boolean_Not, Boolean_Or, Boolean_Equals, Boolean_Not_Equals, Ask_OK_Cancel, Ask_Yes_No
**Boolean_Equal**

Compare the two parameters and return TRUE if they are both TRUE or if they are both FALSE. If the parameters do not have the same value, return FALSE. Use Boolean variables as the parameters instead of actions to make the script easier to read and understand.

**Parameters**

- Test1 A Boolean variable, constant, or action.
- Test2 A Boolean variable, constant, or action.

**Format**

```
Boolean_Equal (Test1, Test2)
```

**Return Value**

Returns a Boolean. TRUE if both Test1 and Test2 are TRUE, or both Test1 and Test2 are FALSE. Returns FALSE otherwise.

**Example**

```
Script(Boolean_Equal_Test)
Boolean(bResultAll)
Boolean(bResult1)
Boolean(bResult2)
Activate(From_Menu)
   bResult1 = Ask_Yes_No("Hit Yes in each message box", "Message 1", FALSE)
   bResult2 = Ask_Yes_No("Hit Yes again", "Message 2", FALSE)
   bResultAll = Boolean_Equal(bResult1, bResult2)
   If(bResultAll)
      Message("Both responses were the same", 5)
   Else
      Message("The responses were different", 5)
   End_If
Return
```

**See Also**

- Boolean_Not_Equal
- Boolean_Set
- Boolean_Not
- Boolean_And
- Boolean_Or
- Ask_OK_Cancel
- Ask_Yes_No
Chapter 7: Overview of Actions

**Boolean_Not**

Set the value of a Boolean variable to FALSE if the parameter is TRUE. Set the value of a Boolean variable to TRUE if the parameter is FALSE.

**Parameters**

*Test*  May be a Boolean action, variable, or constraint.

**Format**

Boolean_Not (Test)

**Return Value**

Returns a Boolean. FALSE if the test is TRUE, returns TRUE otherwise.

**Example**

```plaintext
Script(Boolean_Not_Test)
Boolean(bResult)
Activate(From_Menu)
    bResult = Ask_OK_Cancel("OK for FALSE, Cancel for TRUE", "Boolean_Not_Test", FALSE)
    bResult = Boolean_Not(bResult)
    If(bResult)
        Message("bResult is TRUE", 5)
    Else
        Message("bResult is FALSE", 5)
    End_If
Return
```

**See Also**

Boolean_Set, Boolean_And, Boolean_Or, Boolean_Equal, Boolean_Not_Equal, Ask_OK_Cancel, Ask_Yes_No
**Boolean_Not_Equal**

Compare the two parameters and return TRUE if they do not have the same value. Use Boolean variables as the parameters instead of actions to make the script easier to read and understand.

**Parameters**

- **Test1** A Boolean variable, constant, or action.
- **Test2** A Boolean variable, constant, or action.

**Format**

```
Boolean_Not_Equal (Test1, Test2)
```

**Return Value**

Returns a Boolean. FALSE if both Test1 and Test2 are TRUE, or both Test1 and Test2 are FALSE. Returns TRUE otherwise.

**Example**

```plaintext
Script(Boolean_Not_Equal_Test)
Boolean(bResultAll)
Boolean(bResult1)
Boolean(bResult2)
Activate(From_Menu)
   bResult1 = Ask_Yes_No("Hit Yes in one message box", "Message 1", FALSE)
   bResult2 = Ask_Yes_No("Hit No if you hit Yes in the last message box", "Message 2", FALSE)
   bResultAll = Boolean_Not_Equal(bResult1, bResult2)
   If(bResultAll)
      Message("The responses were different", 5)
   Else
      Message("Both responses were the same", 5)
   End_If
Return
```

**See Also**

Boolean_Equal, Boolean_Set, Boolean_Not, Boolean_And, Boolean_Or, Ask_OK_Cancel, Ask_Yes_No
**Boolean\_Or**

Test each of the parameters and return TRUE if one or more parameters are TRUE. Return FALSE if all parameters are FALSE. One to five parameters may be used for this action.

All tests will be evaluated each time this action is taken. Use Boolean variables as the parameters instead of actions to make the script easier to read and understand.

---

### Parameters

- **Test1** A Boolean variable, constant, or action.
- **Test2** An optional Boolean variable, constant, or action.
- **Test3** An optional Boolean variable, constant, or action.
- **Test4** An optional Boolean variable, constant, or action.
- **Test5** An optional Boolean variable, constant, or action.

---

### Format

Boolean\_Or (Test1, Test2, Test 3, etc.)

---

### Return Value

Returns a Boolean. TRUE if one or more test values are TRUE. Returns FALSE otherwise.

---

### Example

Script(Boolean\_Or\_Test)
Boolean(bResultAll)
Boolean(bResult1)
Boolean(bResult2)
Activate(From\_Menu)
    bResult1 = Ask\_OK\_Cancel("Hit OK in one message box", "Message 1", FALSE)
    bResult2 = Ask\_OK\_Cancel("Hit OK if you hit Cancel in the last box", "Message 2", FALSE)
    bResultAll = Boolean\_Or(bResult1, bResult2)
If(bResultAll)
    Message("bResultAll is TRUE", 5)
Else
    Message("bResultAll is FALSE", 5)
End\_If
Return
See Also

Boolean_Set, Boolean_Not, Boolean_And, Boolean_Equal, Boolean_Not_Equal, Ask_OK_Cancel, Ask_Yes_No
Boolean_Set

Set the value of a Boolean variable. A typical use of Boolean_Set is to set a variable to the return value of another action. The equal sign (=) is the symbol for Boolean_Set in the Script Editor.

Parameters

Test  May be a Boolean action, variable, or constraint.

Format

Boolean_Set (Test)

Return Value

Returns a Boolean. TRUE if the test is TRUE, returns FALSE otherwise.

Example

Script(Boolean_Set_Test)
Boolean(bResult)
Activate(From_Menu)
   bResult = Ask_OK_Cancel("OK for TRUE, Cancel for FALSE", "Boolean_Set_Test", FALSE)
   If(bResult)
      Message("bResult is TRUE", 5)
   Else
      Message("bResult is FALSE", 5)
   End_If
Return

See Also

Boolean_Not, Boolean_And, Boolean_Or, Boolean_Equal, Boolean_Not_Equal, Ask_OK_Cancel, Ask_Yes_No
Break

Jumps to the first statement following the next EndWhile statement (exiting the loop). In a Case statement, jumps to the first statement following the End_Switch statement.

This command is only valid inside of a While loop or a Case statement.

Example #1

Script(Continue_Test)
  Boolean(bContinue)
  Activate(From_Menu)
    While(TRUE)
      bContinue = Ask_OK_Cancel("Press OK to keep getting this message.", "Test", FALSE)
      If(bContinue)
        Continue
      Else
        Break
      End_If
    End_While
  Return

Example #2

strEntered = Ask_String("Type a string", "Switch_Test", 0, 0, "")
Switch( strEntered )
  Case( "Hi" )
    Ask_Ok("Hi", "Switch Result")
    Break
  Case( "Bye" )
    Ask_Ok("Bye", "Switch Result")
    Break
  Case( "OK" )
  Case( "Ok" )
  Case( "ok" )
    Ask_Ok("OK", "Switch Result")
    Break
  Default
    Ask_Ok("Default action", "Switch Result")
    Break
  End_Switch
Return
See Also

While, While_Not, End_While, Continue, Switch, Case, Default, End_Switch
Chapter 7: Overview of Actions

Button(Bitmap_Create_Emulation)

Creates a button with the specified bitmap name, and puts the left side of it where emulation text at the supplied coordinates would be.

Each time the button is pressed, the Boolean variable specified will be set to TRUE. You will need to reset the variable if you want to detect future button presses. All buttons created by the script will be removed when the script exits. The Wait_For_Screen_Update action can be used to wait for a button to be pressed.

The button will be hidden if the emulation text at that location is hidden.

You can add bitmaps to the resource file by using the Resource Editor. For information about the Resource Editor, see Wavelink Terminal Emulation Client User Guide. Row-1 is the top line of text on the screen; column-1 is the left-most column of text on the screen.

Parameters

*Bitmap Name*  The name of the bitmap.
*Row*  The top of the bitmap starts in this text row.
*Column*  The left side of the bitmap starts in this text column.
*Pressed*  Indicates whether the button has been pressed by the user.

Format

```
Button(Bitmap_Create_Emulation ("Bitmap Name", Row, Column, Pressed))
```

Example

```
Script( Button_Bitmap )
Boolean( Pressed )
Activate( From_Menu )
    Button(Bitmap_Create_Emulation( "GOCONTROL", 6, 17, Pressed ))
While_Not( Pressed )
    Wait_For_Screen_Update
End_While
Button_Remove( "GOCONTROL" )
Pressed = FALSE
Return
```
See Also

Wait_For_Screen_Update, Button_Bitmap_Create_View, Button_Create_Emulation, Button_Create_View, Button_Remove, Button_Remove_All
Button_Bitmap_Create_View

Creates a button with the specified bitmap name. This command is the same as Button_Bitmap_Create_Emulation, except that the screen position is used instead of text position, allowing the button to always be visible.

If Button_Bitmap_Create_View is used to create a button at position 1,1, that button will always be in the upper-left corner of the Terminal Emulation view screen. A Button_Bitmap_Create_Emulation button will be hidden if the emulation text at that location is hidden.

A bottom and/or right value of 1000 represents the bottom or right side of the screen. For example, a button at position 1,990 would start 11 columns left of the upper-right corner of the screen.

Each time the button is pressed, the Boolean variable specified will be set to TRUE. You will need to reset the variable if you want to detect future button presses. The Wait_For_Screen_Update action can be used to wait for a button to be pressed. All buttons created by the script will be removed when the script exits.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bitmap Name</td>
<td>The name of the bitmap.</td>
</tr>
<tr>
<td>Row</td>
<td>The top of the bitmap starts in this text row.</td>
</tr>
<tr>
<td>Column</td>
<td>The left side of the bitmap starts in this text column.</td>
</tr>
<tr>
<td>Pressed</td>
<td>Indicates whether the button has been pressed by the user.</td>
</tr>
</tbody>
</table>

Format

Button_Bitmap_Create_View ("Bitmap Name", Row, Column, Pressed)

Example

Script( Button_Bitmap_Create_View_Test )
Boolean( Pressed )
Activate( From_Menu )
    Button_Bitmap_Create_View( "STOPCONTROL", 1000, 1000, Pressed )
    While_Not( Pressed )
        Wait_For_Screen_Update
    End_While
Button_Remove_All
Return
See Also

Button_Bitmap_Create_Emulation, Button_Create_Emulation, Button_Create_View, Button_Remove, Button_Remove_All
**Button_Create_Emulation**

Creates a button with the specified text and puts the left side of it where emulation text at the supplied coordinates would be.

If the width value is 0, the button will be sized to fit the text. Each time the button is pressed, the Boolean variable specified will be set to TRUE. You will need to reset the variable if you want to detect future button presses. All buttons created by the script will be removed when the script exits. The `Wait_For_Screen_Update` action can be used to wait for a button to be pressed.

The button will be hidden if emulation text at that location is hidden.

**Parameters**

- **Text** The text displayed in the button.
- **Row** The top of the bitmap starts in this text row.
- **Column** The left side of the bitmap starts in this text column.
- **Width** The number of characters in the button text.
- **Pressed** Indicates whether the button has been pressed by the user.

**Format**

```
Button_Create_Emulation ("Text", Row, Column, Width, Pressed)
```

**Example**

```plaintext
Script( Button_Create_Emulation_Test )
Boolean( Pressed )
Activate( From_Menu )
    Button_Create_Emulation( "Emulation Button", 2, 1, 0, Pressed )
    While_Not( Pressed )
        Wait_For_Screen_Update
    End_While
Button_Remove( "Emulation Button" )
Pressed = FALSE
Return
```

**See Also**

`Wait_For_Screen_Update, Button_Create_View, Button_Remove, Button_Remove_All, Button_Bitmap_Create_Emulation, Button_Bitmap_Create_View`
**Button_Create_View**

Creates a button with the specified text. This command is the same as **Button_Create_Emulation** except that the screen position is used instead of the text position, allowing the button to always be visible.

If **Button_Create_View** is used to create a button at position 1,1, that button will always be in the upper-left corner of the Terminal Emulation view screen. A **Button_Create_Emulation** button will be hidden if the emulation text at that location is hidden. A bottom and/or right value of 1000 represents the bottom or right side of the screen. For example, a button at position 1,900 would start 11 columns left of the upper-right corner of the screen.

Each time the button is pressed, the Boolean variable specified will be set to TRUE. You will need to reset the variable if you want to detect future button presses. The **Wait_For_Screen_Update** action can be used to wait for a button to be pressed. All buttons created by the script will be removed when the script exits.

---

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Text</strong></td>
<td>The text displayed in the button.</td>
</tr>
<tr>
<td><strong>Row</strong></td>
<td>The top of the bitmap starts in this text row.</td>
</tr>
<tr>
<td><strong>Column</strong></td>
<td>The left side of the bitmap starts in this text column.</td>
</tr>
<tr>
<td><strong>Width</strong></td>
<td>The number of characters in the button text.</td>
</tr>
<tr>
<td><strong>Pressed</strong></td>
<td>Indicates whether the button has been pressed by the user.</td>
</tr>
</tbody>
</table>

---

**Format**

```
Button_Create_View ("Text", Row, Column, Width, Pressed)
```

---

**Example**

```plaintext
Script( Button_Create_View_Test )
Boolean( Pressed )
Activate( From_Menu )
    Button_Create_View( "This is a button", 1, 1, 0, Pressed )
    While_Not( Pressed )
        Wait_For_Screen_Update
    End_While
End_While
Button_Remove_All
Return
```
See Also

Button_Create_Emulation, Button_Remove, Button_Remove_All, Button_Bitmap_Create_Emulation, Button_Bitmap_Create_View
Button_Remove

Removes a button created with the Button_Create_Emulation and Button_Create_View actions with the specified text.

Parameters

Text  The text displayed in the button.

Format

Button_Remove ("Text")

Example

Script( Button_Remove_Test )
Boolean( Pressed )
Activate( From_Menu )
    Button_Create_View( "This is a button", 1, 1, 0, Pressed )
While_Not( Pressed )
    Wait_For_Screen_Update
End_While
Button_Remove( "This is a button" )
Ask_OK( "The button was removed.", "Button_Remove_Test" )
Return

See Also

Button_Create_Emulation, Button_Create_View, Button_Remove_All
**Button_Remove_All**

Removes all buttons created with the `Button_Create_Emulation` and `Button_Create_View` action for this script.

**Example**

```wscript
Script( Button_Remove_All_Test )
Boolean( Pressed )
Activate( From_Menu )
    Button_Create_View( "This is a button", 1, 1, 0, Pressed )
    Button_Create_View( "Another button", 4, 1, 0, Pressed )
    While_Not( Pressed )
        Wait_For_Screen_Update
    End_While
    Button_Remove_All
    Ask_OK( "All buttons removed.", "Button_Remove_All_Test" )
Return
```

**See Also**

`Button_Create_Emulation`, `Button_Create_View`, `Button_Remove`, `Button_Bitmap_Create_Emulation`, `Button_Bitmap_Create_View`
Call

Suspended the current script and executes another script. The current script resumes when the called script exits. The name of the script is case-sensitive.

There are two ways to use the Call function. The first way allows passing an unlimited number of arguments to the called script. The second way allows you to pass up to four arguments to the called script and allows you to return a value. See the examples below for the format to use when passing arguments to a called script.

Parameters

<table>
<thead>
<tr>
<th>Script Name</th>
<th>The name of the script to call.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arg*</td>
<td>Arguments to pass to the called script.</td>
</tr>
</tbody>
</table>

Format

See below.

Example #1

The snippet of code below demonstrates a call to another script and how to pass arguments back and forth between scripts. On the lines following the Call statement, the variables on the left are in the called script while the variables on the right are in the calling script. The \(<---\) means that data from the calling script will be passed to the called script variables but not returned to the calling script. The \(\leftrightarrow\) means that data from the calling script will be passed to the called script variables on the left and the called script will return values to the variables found in the calling script.

Comment: Call sapFieldEntry to update the field on the web page.

If ( bVerifySpeechValue )
    Comment: Assign value to appropriate field
    Call: sapFieldEntry
        strFieldName \(<---\) "s4000_binp[1]"
        strFieldValue \(<---\) strPromptBinValue
    Speech_To_Text_Cancel
    Break
Else
    Call: sapSpeechEntryBin
        strPromptBin \(<---\) strPromptBin
        strPromptBinValue \(<---\) strPromptBinValueStripped
        bVerifySpeech \(<---\) bVerifySpeechValue
        bRepeatPrompts \(<---\) bRepeatPrompts
        bBackFunction \(<---\) bBackFunction
Example #2

The first script callFunctionSample calls the scripts callConcatenateStrings and callSumsTotal, passing them parameters and using the return values from the scripts. All three scripts are included in this example.

callFunctionSample

Script( callFunctionSample )
String( strEntry )
String( strArg1 )
String( strArg2 )
String( strArg3 )
String( strArg4 )
String( strResultString )
Number( nTotalSum )
Number( nIndex )
Number( nArg1 )
Activate( From_Menu )

    nArg1 = Ask_Number( "Enter a Number:", "Sum Multiple Numbers", 1, 200, 25 )

    nTotalSum = Call( "callSumTotal", nArg1, Number_Plus( nArg1, 5 ),
Number_Plus( nArg1, 10 ), Number_Plus( nArg1, 20 ) )
    strResultString = String_Combine( "The sum is: ", Number_To_String_Decimal( nTotalSum ) )
    Ask_OK( strResultString, "Sum Result" )
    strResultString = ""

    nIndex = 0
    While( Number_Less_Than( nIndex, 4 ) )
        strEntry = Ask_String( "Enter String:", "Concatenate strings", 1, 20, "hello" )

        Switch( nIndex )
        Case( 0 )
            strArg1 = strEntry
            Break
        Case( 1 )
            strArg2 = strEntry
            Break
        Case( 2 )
strArg3 = strEntry
Break
Case( 3 )
    strArg4 = strEntry
    Break
End_Switch

nIndex = Number_Plus( nIndex, 1 )
strEntry = ""
End_While

strResultString = Call( "callConcatenateStrings", strArg1, strArg2, strArg3, strArg4 )
    Ask_OK( strResultString, "String Concatenation" )
Return

callSumTotal

Script( callSumTotal )
Parameter( Number, nArg1 )
Parameter( Number, nArg2 )
Parameter( Number, nArg3 )
Parameter( Number, nArg4 )
Activate( From_Other_Script )

    nArg1 = Number_Plus( nArg1, nArg2 )
    nArg1 = Number_Plus( nArg1, nArg3 )
    nArg1 = Number_Plus( nArg1, nArg4 )

Return( nArg1 )

callConcatenateStrings

Script( callConcatenateStrings )
Parameter( String, strCat1 )
Parameter( String, strCat2 )
Parameter( String, strCat3 )
Parameter( String, strCat4 )
Activate( From_Other_Script )

    strCat1 = String.Combine( strCat1, strCat2, strCat3, strCat4 )
    strCat1 = String.Combine( "The result is: ", strCat1 )

Return( strCat1 )
See Also

Abort, Abort_All, Return
Chapter 7: Overview of Actions

Cancel_Other_Scripts

Cancels all other scripts for the session with the script name. If the name is left blank, all other scripts for the session are canceled.

The calling script is never canceled.

Parameters

Value1  This is the script name (a string).

Format

Cancel_Other_Scripts("Value1")

Example

Script( Cancel_Script_All )
String( sMessage )
Number( nScriptsCancelled )
Activate( From_Menu )
    nScriptsCancelled = Cancel_Other_Scripts( "" )
    sMessage = String_Combine( Number_To_String_Decimal( nScriptsCancelled ), " script(s) cancelled." )
    Ask_OK( sMessage, "Results" )
Return
**Case**

If the Test parameter matches the value of the previous Switch parameter, the script continues executing until the next Break or End_Switch statement.

Otherwise, the next Case action is executed to check if its Test matches the Switch parameter value. If none of the Case parameters match the Switch parameter, then the Default action is executed.

This command is only valid inside a Case statement. Test may be a constant or variable String or Number.

**Parameters**

- **Test**
  
  If the value matches the value of the previous Switch parameter, then the next set of actions gets executed, up to the Break or End_Switch, whichever comes first.

**Format**

```
Case( Test )
```

**Example**

```wbs
Script(Switch_Test)
String(strEntered)
Activate(From_Menu)

strEntered = Ask_String("Type a string", "Switch_Test", 0, 0, "")
Switch( strEntered )
Case( "Hi" )
  Ask_Ok("Hi", "Switch Result")
  Break
Case( "Bye" )
  Ask_Ok("Bye", "Switch Result")
  Break
Case( "OK" )
Case( "Ok" )
Case( "ok" )
  Ask_Ok("OK", "Switch Result")
  Break
Default
  Ask_Ok("Default action", "Switch Result")
  Break
End_Switch
Return
```
See Also

Switch, Break, Default, End_Switch
Chapter 7: Overview of Actions

Character_To_Number

Converts the character at position Index in the string into the number value for that character. An index of 0 indicates the left-most character in the string.

Parameters

- **String**  The string containing the conversion character.
- **Index**  The index of the character in the string.

Format

Character_To_Number (String, Index)

Return Value

Returns a character’s number value. If the index does not point to a character, a value of 0 is returned.

Example

```wawk
Script(Character_To_Number_Test)
String(strCharacters)
String(strTitle)
Number(nIndex)
Number(nCharacter)
Number(nToConvert)
Activate(From_Menu)
    strCharacters = Ask_String("Enter a string", "Character_To_Number", 1, 99, "abcde")
    nIndex = Ask_Number("Enter the index of the character to convert to a number", "Character_To_Number", 0, 99, 0)
    nCharacter = Character_To_Number(strCharacters, nIndex)
    strTitle = String_Combine("", strTitle)
    strTitle = String_Combine(Number_To_Character(nCharacter), strTitle)
    strTitle = String_Combine("Character_To_Number of ", strTitle)
    Ask_OK(Number_To_String_Decimal(nCharacter), strTitle)
Return
```

See Also

- Ask_String, Search_Screen, Speech_To_Text, Get_Screen_Text
Comment

Proceeds to the next instruction without taking any action.

Parameters

Comment: Text that describes the script.

Format

Comment: (Comment)

Example

Script(Comment_Test)
  Activate(From_Menu)
    Comment: This script pops up a message, for testing.
    Message("Testing...", 0)
  Return

See Also

Blank_Line
Chapter 7: Overview of Actions

Continue

Jumps back to the last While statement and re-tests the test value. This action is only valid inside of a While loop.

Example

Script(Continue_Test)
  Boolean(bContinue)
  Activate(From_Menu)
    While(TRUE)
      bContinue = Ask_OK_Cancel("Press OK to keep getting this message.", "Test", FALSE)
      If(bContinue)
        Continue
      Else
        Break
      End_If
    End_While
  Return

See Also

While, While_Not, End_While, Break
Default

If none of the Case parameter values match the value of the previous Switch parameter, the script continues executing until the next Break or End_Switch statement. Otherwise, jumps to the first statement following the next End_Switch statement. This command is only valid inside a Case statement.

Example

```plaintext
Script(Switch_Test)
String(strEntered)
Activate(From_Menu)
strEntered = Ask_String("Type a string", "Switch_Test", 0, 0, ")
Switch( strEntered )
Case( "Hi" )
    Ask_Ok("Hi", "Switch Result")
    Break
Case( "Bye" )
    Ask_Ok("Bye", "Switch Result")
    Break
Case( "OK" )
Case( "Ok" )
Case( "ok" )
    Ask_Ok("OK", "Switch Result")
    Break
Default
    Ask_Ok("Default action", "Switch Result")
    Break
End_Switch
Return
```

See Also

Switch, Case, Break, End_Switch
Delay

Suspends the current script until the specified time has passed.

Parameters

\[ Time \ (\text{Milliseconds}) \]  The duration of the delay.

Format

Delay (Time)

Example

Script(Delay_Test)
   Activate(From_Menu)
   Message("Showing a message for a few seconds.", 0)
   Delay(4000)
   Message_Clear
   Return

See Also

Wait_For_Screen_Update, Wait_For_Screen_Update_With_Timeout, Get_Time
Disconnect

Exits all scripts for the session and disconnects the session.

Example

Script(Disconnect_Test)
    Activate(From_Menu)
    Comment: This script causes all of the session’s scripts to end and the session to disconnect.
    Disconnect

See Also

Return, Abort, Abort_All, Exit_Application, Reboot, Abort, Abort_All, Exit_Application, Suspend
Else

Start of statements to be executed if an If test fails. This command is only valid inside of an If block.

Example

Script(If_Else_Test)
    Boolean(bOK)
    Activate(From_Menu)
        bOK = Ask_OK_Cancel("Press OK or Cancel.", "Test", FALSE)
        If(bOK)
            Message("OK is TRUE", 0)
        Else
            Message("OK is FALSE", 0)
        End_If
    Return

See Also

If, If_Not, End_If
End_If

End of statements to be executed for an If test.

Example

Script( If_Test )
Boolean( bOK )
Activate( From_Menu )
    Message_Clear
    bOK = Ask_OK_Cancel( "Press OK to see another message.", "Press OK", FALSE )
    If( bOK )
        Message( "bOK is TRUE", 0 )
    End_If
Return

See Also

If, If_Not, Else
End_Switch

End of statements to be executed for a Switch test.

Example

```plaintext
Script(Switch_Test)
String(strEntered)
Activate(From_Menu)
strEntered = Ask_String("Type a string", "Switch_Test", 0, 0, "")
Switch( strEntered )
Case( "Hi" )
    Ask_Ok("Hi", "Switch Result")
    Break
Case( "Bye" )
    Ask_Ok("Bye", "Switch Result")
    Break
Case( "OK" )
Case( "Ok" )
Case( "ok" )
    Ask_Ok("OK", "Switch Result")
    Break
Default
    Ask_Ok("Default action", "Switch Result")
    Break
End_Switch
Return
```

See Also

Switch, Case, Default, Break
End_While

End of statements to be executed for a While test.

Example

Script(End_While_Test)
  Boolean(bOK)
  Activate(From_Menu)
  bOK = TRUE
  While(bOK)
    bOK = Ask_OK_Cancel("Press OK to keep getting this message.", "Test", FALSE)
  End_While
  Return

See Also

While, While_Not, Continue, Break
**Escape_Sequence**

Handles the supplied Wavelink Custom or Telxon ESC Sequence for all emulation types. The sequence should be all the characters that will follow the first ESC character.

**Parameters**

*Sequence*  The escape sequence.

**Format**

```
Escape_Sequence (Sequence)
```

**Return Value**

The string returned will be the sequence returned by the ESC sequence (without the initial ESC) or an empty string if the sequence returns nothing.

**Example**

```plaintext
Script(Quite_Mode_Escape_Sequence)
String(sResult)
Activate(From_Menu)
    sResult = Escape_Sequence("%2Q")
Ask_OK(sResult, "Quiet Mode ESC Sequence Result")
Return
```
Chapter 7: Overview of Actions

Exit_Application

Shuts down the Terminal Emulation application.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return Value</td>
<td>The value that the Terminal Emulation application returns to the system as it exits.</td>
</tr>
</tbody>
</table>

Format

Exit_Application (Return Value)

Return Values

The return value is the application exit value Terminal Emulation will use when exiting.

Example

    Script(Exit_Application_Test)
    Activate(From_Menu)
        Comment: this script shuts down the Terminal Emulation application.
        Exit_Application(1)

See Also

Return, Abort, Abort_All, Disconnect, Reboot
Get_Avalanche_Property_Value

Returns a string with the value of the Wavelink Avalanche property. Returns an empty string if Avalanche is not installed or does not have that property.

Parameters

Value1 The property name (a string).

Format

Get_Avalanche_Property_Value(Value1)

Example

Script( Avalanche_Property )
String( sName, True )
String( sResult )
Activate( From_Menu )
   sName = Ask_String( "What is the name of the Avalanche property?", "Avalanche Property Test", 1, 200, sName )
   If_Not( String_Empty( sName ) )
      sResult = Get_Avalanche_Property_Value( sName )
      If_Not( String_Empty( sResult ) )
         Ask_OK( String_Combine( "The property value is ", sResult ), "Result" )
      Else
         Ask_OK( "No property value available.", "Result" )
   End_If
End_If
Return
Chapter 7: Overview of Actions

Get_Field_Append_Scan_Data

Query whether data is appended when the field is scanned. This action is only valid when using IBM 5250 or 5555 emulation.

Parameters

*Field Index*  
The numeric index of the 5250 data field, index 0 is the first field.

Format

Get_Field_Append_Scan_Data (Field Index)

Example

```c
Script(Get_Field_Append_Scan_Data_Test)
Boolean(bAppend)
Activate(From_Menu)
    bAppend = Get_Field_Append_Scan_Data(0)
    If(bAppend)
        Message("Appending scan data", 5)
    Else
        Message("Not appending scan data", 5)
    End_If
Return
```

See Also

Set_Field_Data_ID, Set_Field_Symbology_ID, Get_Field_SymbologyOperator, Set_Field_Append_Scan_Data, Set_Field_Com_Data_Field, Set_Field_Prefix_Scan_Data, Get_Field_Data_ID, Get_Num_Field_Data_IDs, Get_Num_Field_Symbology_IDs, Get_Field_Com_Data_Field, Get_Field_Symbology_ID, Get_Num_Fields, Get_Field_Prefix_Scan_Data
Get_Field_Column

Get the column number of the field.

Parameters

Field Index  The index of the field.

Format

Get_Field_Column (Field Index)

Example

Script(Get_Field_Row_Column_Length)
String(strMessage)
Number(numFields)
Number(nLoops)
Number(nFieldRow)
Number(nFieldColumn)
Number(nFieldLength)
Activate(From_Menu)
    numFields = Get_Num_Fields
    Message(String_Combine("Number of fields:", Number_To_String_
Decimal(numFields)), 60)
    nLoops = 0
    While(Number_Less_THAN(nLoops, numFields))
        nFieldRow = Get_Field_Row(nLoops)
        nFieldColumn = Get_Field_Column(nLoops)
        nFieldLength = Get_Field_Length(nLoops)
        strMessage = String_Combine("Field:", Number_To_String_
Decimal(nLoops))
        strMessage = String_Combine(strMessage, ":row")
        strMessage = String_Combine(strMessage, Number_To_String_
Decimal(nFieldRow))
        strMessage = String_Combine(strMessage, ", column")
        strMessage = String_Combine(strMessage, Number_To_String_
Decimal(nFieldColumn))
        strMessage = String_Combine(strMessage, ", length")
        strMessage = String_Combine(strMessage, Number_To_String_
Decimal(nFieldLength))
        Ask_OK(strMessage, "Field Info")
        nLoops = Number_Plus(nLoops, 1)
    End_While
Comment: The following should return zero because the field index is invalid.

```plaintext
nFieldRow = Get_Field_Row(nLoops)
nFieldColumn = Get_Field_Column(nLoops)
nFieldLength = Get_Field
Return
```

See Also

Get_Num_Fields, Get_Field_Index_Row_Text, Get_Field_Index_Column_Text, Get_Field_Row, Get_Field_Length, Get_Field_Symbology_Operator, Set_Field_Append_Scan_Data, Set_Field_Com_Data_Field, Set_Field_Prefix_Scan_Data, Get_Field_Append_Scan_Data, Get_Num_Field_Data_IDS, Get_Num_Field_Symbology_IDS, Get_Field_Com_Data_Field, Set_Field_Data_ID, Set_Field_Symbology_ID, Get_Field_Symbology_ID
Get_Field_Com_Data_Field

Get the index of the field that is the Com data field. An index of \(-1\) means that no field is the Com data field. This action is only valid when using IBM 5250 or 5555 emulation.

Example

```plaintext
Script(SetGet_Field_Com_Data_Field_Test)
Boolean(bSetOK)
Number(nFieldID)
Activate(From_Menu)
  bSetOK = Set_Field_Com_Data_Field(2, TRUE)
  If(bSetOK)
    nFieldID = Get_Field_Com_Data_Field
    Message(String_Combine("Get_Field_Com_Data_Field: ", Number_To_String_Decimal(nFieldID)), 7)
  Else
    Message("Set_Field_Com_Data_Field failed", 5)
  End_If
Return
```

See Also

Get_Num.Fields, Get_Field_Index_Row_Text, Get_Field_Index_Column_Text, Get_Field_Row, Get_Field_Length, Get_Field_Symbology_OPERATOR, Set_Field_Append_Scan_Data, Set_Field_Com_Data_Field, Set_Field_Prefix_Scan_Data, Get_Field_Append_Scan_Data, Get_Field_Column, Get_Num_Field_Data_IDS, Get_Num_Field_Symbology_IDS, Set_Field_Data_ID, Set_Field_Symbology_ID, Get_Field_Symbology_ID
Get_Field_Data_ID

Gets the data ID for the specified field. A blank string means no Data ID is set for the field or the field index is invalid. A field may have more than one data ID. Use Get_Num_Field_Data_IDs to determine the number of data IDs for a field. This action is only valid when using IBM 5250 or 5555 emulation.

Parameters

- **Field Index**: The index of the field.
- **Data ID Index**: The index of the Data ID.

Format

Get_Field_Data_ID (Field Index, Data ID Index)

Return Value

Returns the field’s Data ID.

Example

```script
Script( Get_Field_Data_ID_Test )
String( strDataID )
Boolean( bSetOK )
Number( numDataIDs )
Number( counter )
Activate( From_Menu )
    bSetOK = Set_Field_Data_ID( 0, "N" )
    numDataIDs = Get_Num_Field_Data_IDs( 0 )
    counter = 0
    While( Number_Less_Than( counter, numDataIDs ) )
        strDataID = Get_Field_Data_ID( 0, counter )
        Ask_OK( strDataID, "Data ID for Field 0" )
        counter = Number_PLUS( counter, 1 )
    End_While
Return
```

See Also

Get_Field_Prefix_Scan_Data, Get_Num_Field_Data_IDs, Get_Num_Field_Symbology_IDs, Get_Field_Com_Data_Field, Get_Field_Symbology_ID, Get_Num_Fields, Set_Field_Symbology_ID,
Set_Field_Data_ID, Get_Field_Symbology_Operator, Set_Field_Append_Scan_Data, Set_Field_Com_Data_Field, Set_Field_Prefix_Scan_Data, Get_Field_Append_Scan_Data
Get_Field_Index

Get the index of a field at the specified row and column. An index of -1 means there is no field at the row and column. This action is only valid when using IBM 5250 or 5555 emulation.

Parameters

Row        The row containing the field.
Column     A column in the field.

Format

Get_Field_Index (Row, Column)

Example

Script( Get_Field_Index_Test )
String( strMessage )
Number( nRow )
Number( nColumn )
Number( nFieldIndex )
Activate( From_Menu )
    nRow = Ask_Number( "Enter the row number containing the field", "Get_Field_Index", 1, 999, 1 )
    nColumn = Ask_Number( "Enter a column number in the field", "Get_Field_Index", 1, 999, 1 )
    nFieldIndex = Get_Field_Index( nRow, nColumn )
    strMessage = String_Combine( "Field at row ", Number_To_String_Decimal( nRow ) )
    strMessage = String_Combine( strMessage, ", column " )
    strMessage = String_Combine( strMessage, Number_To_String_Decimal( nColumn ) )
    strMessage = String_Combine( strMessage, ": " )
    strMessage = String_Combine( strMessage, Number_To_String_Decimal( nFieldIndex ) )
Message( strMessage, 12 )
Return

See Also

Get_Screen_Columns, Get_Screen_Rows, Get_Position_Column, Get_Position_Row, Get_Session_Number, Get_Time, Get_Time_Since_Reset, Get_Num_Fields, Get_Field_Index_Row_
Text, Get_Field_Index_Column_Text, Get_Field_Row, Get_Field_Column, Get_Field_Length, Get_Num_Field_Data_IDS, Get_Num_Field_Symbology_IDS, Get_Field_Com_Data_Field
Get_Field_Index_Column_Text

Get the index of a field that is in the same column as the text. The text may be above or below the field in the same row as the field. An index of \(-1\) means either the text was not found or there is no field before or after the text in the column where the text was found. This action is only valid when using IBM 5250 or 5555 emulation.

**Parameters**

- **Screen Text**
  The text in the same column as the field.
- **Text Above Field**
  Indicates whether the text is above or below the field.
- **Ignore Case**
  Indicates whether the case of the letters is taken into consideration.

**Format**

Get_Field_Index_Column_Text (Screen Text, Text Above Field, Ignore Case)

**Return Value**

Returns a Boolean. TRUE if the text is above the field, FALSE if the text is below the field.

**Example**

```plaintext
Script(Get_Field_Index_Column_Text_Test)
String(strTextInColumn)
Boolean(bTextAboveField)
Number(nFieldIndex)
Activate(From_Menu)
    strTextInColumn = Ask_String("Enter some text in the same column as the field", "Get_Field_Index_Column_Text", 1, 99, ")
    bTextAboveField = Ask_Yes_No("Is the text above the field?", "Get_Field_Index_Column_Text", FALSE)
    nFieldIndex = Get_Field_Index_Column_Text(strTextInColumn, bTextAboveField, FALSE)
    Message(String_Combine("Field ID (0 is first field): ", Number_To_String_Decimal(nFieldIndex)), 5)
Return
```

**See Also**

Get_Num_Fields, Get_Field_Index_Row_Text, Get_Field_Row, Get_Field_Length, Get_Field_Symbology_Operator, Set_Field_Append_Scan_Data, Set_Field_Com_Data_Field, Set_Field_
Prefix_Scan_Data, Get_Field_Append_Scan_Data, Get_Field_Column, Get_Num_Field_Data_IDs, Get_Num_Field_Symbology_IDs, Get_Field_Com_Data_Field, Set_Field_Data_ID, Set_Field_Symbology_ID, Get_Field_Symbology_ID
**Get_Field_Index_Row_Text**

Get the index of a field that is in the same row as the text. The text may be before or after the field in the same row as the field. An index of -1 means either the text was not found or there is no field before or after the text in the row where the text was found. This action is only valid when using IBM 5250 or 5555 emulation.

**Parameters**

- **Screen Text**
  - The text on the same row as the field.
- **Text Before Field**
  - Indicates whether the text is before or after the field.
- **Ignore Case**
  - Indicates whether the case of the letters is taken into consideration.

**Format**

```plaintext
Get_Field_Index_Row_Text (Screen Text, Text Before Field, Ignore Case)
```

**Return Value**

Returns a Boolean. TRUE if the text is before the field, FALSE if the text is after the field.

**Example**

```plaintext
Script(Get_Field_Index_Row_Text_Test)
String(strTextInRow)
Boolean(bTextBeforeField)
Number(nFieldIndex)
Activate(From_Menu)
  strTextInRow = Ask_String("Enter some text on the same row as the field", "Get_Field_Index_Row_Text", 1, 99, ")
  bTextBeforeField = Ask_Yes_No("Is the text before the field?", "Get_Field_Index_Row_Text", FALSE)
  nFieldIndex = Get_Field_Index_Row_Text(strTextInRow, bTextBeforeField, FALSE)
  Message(String_Combine("Field ID (0 is first field): ", Number_To_String_Decimal(nFieldIndex)), 5)
Return
```

**See Also**

- Get_Num_Fields, Get_Field_Index_Column_Text, Get_Field_Row, Get_Field_Length, Get_Field_Symbology_Operator, Set_Field_Append_Scan_Data, Set_Field_Com_Data_Field, Set_
Field_Prefix_Scan_Data, Get_Field_Append_Scan_Data, Get_Field_Column, Get_Num_Field_Data_IDs, Get_Num_Field_Symbology_IDs, Get_Field_Com_Data_Field, Set_Field_Data_ID, Set_Field_Symbology_ID, Get_Field_Symbology_ID
Get_Field_Length

Get the length of the field.

Parameters

Field Index  The index of the field.

Format

Get_Field_Length (Field Index)

Example

Script(Get_Field_Row_Column_Length)
String(strMessage)
Number(numFields)
Number(nLoops)
Number(nFieldRow)
Number(nFieldColumn)
Number(nFieldLength)
Activate(From_Menu)
    numFields = Get_Num_Fields
    Message(String_Combine("Number of fields:", Number_To_String__Decimal(numFields)), 60)
    nLoops = 0
    While(Number_Less_Than(nLoops, numFields))
        nFieldRow = Get_Field_Row(nLoops)
        nFieldColumn = Get_Field_Column(nLoops)
        nFieldLength = Get_Field_Length(nLoops)
        strMessage = String_Combine("Field:", Number_To_String__Decimal(nLoops))
        strMessage = String_Combine(strMessage, ":row")
        strMessage = String_Combine(strMessage, Number_To_String__Decimal(nFieldRow))
        strMessage = String_Combine(strMessage, ", column")
        strMessage = String_Combine(strMessage, Number_To_String__Decimal(nFieldColumn))
        strMessage = String_Combine(strMessage, ", length")
        strMessage = String_Combine(strMessage, Number_To_String__Decimal(nFieldLength))
        Ask_OK(strMessage, "Field Info")
        nLoops = Number_PLUS(nLoops, 1)
    End_While
Comment: The following should return zero because the field index is invalid.

\[
\begin{align*}
nFieldRow &= \text{Get\_Field\_Row}(nLoops) \\
nFieldColumn &= \text{Get\_Field\_Column}(nLoops) \\
nFieldLength &= \text{Get\_Field\_Length}(nLoops)
\end{align*}
\]

Return

**See Also**

Get\_Num\_Fields, Get\_Field\_Index\_Row\_Text, Get\_Field\_Index\_Column\_Text, Get\_Field\_Row, Get\_Field\_Symbology\_Operator, Set\_Field\_Append\_Scan\_Data, Set\_Field\_Com\_Data\_Field, Set\_Field\_Prefix\_Scan\_Data, Get\_Field\_Append\_Scan\_Data, Get\_Field\_Column, Get\_Num\_Field\_Data\_IDs, Get\_Num\_Field\_Symbology\_IDs, Get\_Field\_Com\_Data\_Field, Set\_Field\_Data\_ID, Set\_Field\_Symbology\_ID, Get\_Field\_Symbology\_ID
Get_Field_Prefix_Scan_Data

Gets the prefixed data for the specified field. This action is only valid when using IBM 5250 or 5555 emulation.

Parameters

Field Index  The index of the field.

Format

Get_Field_Prefix_Scan_Data (Field Index)

Return Value

Returns the data prefixed when the field is scanned.

Example

Script(Get_Field_Prefix_Scan_Data_Test)
String(strPrefix)
Boolean(bOK)
Activate(From_Menu)
    bOK = Set_Field_Prefix_Scan_Data(0, "SCAN")
    strPrefix = Get_Field_Prefix_Scan_Data(0)
    Ask_OK(strPrefix, "String that will be prefixed to scan data in field")
    Return

See Also

Get_Field_Data_ID, Get_Field_Prefix_Scan_Data, Get_Num_Field_Data_IDS, Get_Num_Field_Symbology_IDS, Get_Field_Com_Data_Field, Get_Field_Symbology_ID, Get_Num_Fields, Set_Field_Symbology_ID, Set_Field_Data_ID, Get_Field_Symbology_Operator, Set_Field_Append_Scan_Data, Set_Field_Com_Data_Field, Set_Field_Prefix_Scan_Data, Get_Field_Append_Scan_Data
Get_Field_Row

Get the row number of the field.

Parameters

Field Index  The index of the field.

Format

Get_Field_Row (Field Index)

Example

Script(Get_Field_Row_Column_Length)
String(strMessage)
Number(numFields)
Number(nLoops)
Number(nFieldRow)
Number(nFieldColumn)
Number(nFieldLength)
Activate(From_Menu)
    numFields = Get_Num_Fields
    Message(String_Combine("Number of fields:", Number_To_String_Decimal(numFields)), 60)
    nLoops = 0
    While(Number_Less_Than(nLoops, numFields))
        nFieldRow = Get_Field_Row(nLoops)
        nFieldColumn = Get_Field_Column(nLoops)
        nFieldLength = Get_Field_Length(nLoops)
        strMessage = String_Combine("Field:", Number_To_String_Decimal(nLoops))
        strMessage = String_Combine(strMessage, ":row")
        strMessage = String_Combine(strMessage, Number_To_String_Decimal(nFieldRow))
        strMessage = String_Combine(strMessage, ":column")
        strMessage = String_Combine(strMessage, Number_To_String_Decimal(nFieldColumn))
        strMessage = String_Combine(strMessage, ":length")
        strMessage = String_Combine(strMessage, Number_To_String_Decimal(nFieldLength))
        Ask_OK(strMessage, "Field Info")
        nLoops = Number_Plus(nLoops, 1)
End_While
Comment: The following should return zero because the field index is invalid.
nFieldRow = Get_Field_Row(nLoops)
nFieldColumn = Get_Field_Column(nLoops)
nFieldLength = Get_Field_Length(nLoops)
Return

See Also

Get_Num_Fields, Get_Field_Index_Row_Text, Get_Field_Index_Column_Text, Get_Field_Length, Get_Field_Symbology_Operator, Set_Field_Append_Scan_Data, Set_Field_Com_Data_Field, Set_Field_Prefix_Scan_Data, Get_Field_Append_Scan_Data, Get_Field_Column, Get_Num_Field_Data_IDS, Get_Num_Field_Symbology_IDS, Get_Field_Com_Data_Field, Set_Field_Data_ID, Set_Field_Symbology_ID, Get_Field_Symbology_ID
Get_Field_Symbolology_ID

Gets the symbolology ID of the specified field.

There may be more than one symbolology ID in a field; pass in the zero-based Symbolology Index. For example, Symbolology Index 0 gets the first symbolology ID. The return ID ANY means Use All Symbolologies. An empty or blank return ID means either the field has no symbolology IDs or the field index is not valid. Use Get_Num_Field_Symbolology_Ids to determine the number of symbolologies for a field. This action is only valid when using IBM 5250 or 5555 emulation.

Scanner symbolology values can be found in Symbologies and Values on page 369.

Parameters

Field Index The index of the field.

Symbolology Index The index of the symbolology.

Format

Get_Field_Symbolology_ID (Field Index, Symbolology Index)

Return Value

Returns the field symbolology ID.

Example

Script(Get_Field_Symbolology_ID_Test)
String(strSymbolologyID)
Boolean(ok)
Number(numSymbolologies)
Number(counter)
Activate(From_Menu)

Comment: Set some symbolologies for field 0, then display them.
ok = Set_Field_Symbolology_ID(0, "UPCE0", FALSE)
ok = Set_Field_Symbolology_ID(0, "CODE 39", FALSE)
ok = Set_Field_Symbolology_ID(0, "EAN8", FALSE)
numSymbolologies = Get_Num_Field_Symbolology_Ids(0)
counter = 0
While(Number_Less_Than(counter, numSymbolologies))
    strSymbolologyID = Get_Field_Symbolology_ID(0, counter)
    Ask_OK(strSymbolologyID, "Symbolology for Field 0")
    counter = Number_Plus(counter, 1)
End_While
Return
See Also

Set_Field_Symbolology_ID, Get_Field_Symbolology_Operator, Get_Field_Data_ID, Get_Num_Field_Symbolology_IDs, Get_Num_Fields
Get_Field_Symbology_Operator

Query whether the field data must match both the field Data ID and the field symbology ID. Use the Set_Field_Symbology_ID And-Or parameter to set whether the field data must match both the field data ID and the field symbology ID.

Parameters

Field Index   The numeric index of the 5250 data field, index 0 is the first field.

Format

Get_Field_Symbology_Operator (Field Index)

Return Value

Returns a Boolean. TRUE if the field data must match both the Data ID and the Symbology ID. Returns FALSE if the field data must match either the Data ID or Symbology ID.

Example

Script( Get_Field_Symbology_Operator_Test )
Boolean( bSetOK )
Boolean( bOperator )
Activate( From_Menu )
   bSetOK = Set_Field_Data_ID( 0, "N" )
   bSetOK = Set_Field_Symbology_ID( 0, "UPCA", TRUE )
   bOperator = Get_Field_Symbology_Operator( 0 )
   If( bOperator )
      Message( "Field data must match both the Data ID and the Symbology ID", 15 )
   Else
      Message( "Field data must match one or the other of Data ID and Symbology ID", 15 )
   End_If
Return

See Also

Set_Field_Data_ID, Set_Field_Symbology_ID, Set_Field_Append_Scan_Data, Set_Field_Com_Data_Field, Set_Field_Prefix_Scan_Data, Get_Field_Append_Scan_Data, Get_Field_Data_ID, Get_Num_Field_Data_IDS, Get_Num_Field_Symbology_IDS, Get_Field_Com_Data_Field, Get_Field_Symbology_ID, Get_Num.Fields, Get_Field_Prefix_Scan_Data
Chapter 7: Overview of Actions

Get_IP_Address

Get the IP address of the device.

Return Value

Returns the current IP address for the device.

Example

Script(Get_IP_Address_Test)
String(stripAddress)
Activate(From_Menu)
    stripAddress = Get_IP_Address
    Ask_OK(stripAddress, "IP Address")
Return

See Also

Get_MAC_Address, Get_Workstation_ID, Get_Session_Number
Get_MAC_Address

Get the MAC address of the device.

Return Value

Returns the current MAC address for the device.

Example

Script(Get_MAC_Address_Test)
String(strMacAddress)
Activate(From_Menu)
    strMacAddress = Get_MAC_Address
    Ask_OK(strMacAddress, "MacAddress")
Return

See Also

Get_IP_Address, Get_Workstation_ID, Get_Session_Number
Get_Num_Field_Data_IDs

Get the number of data IDs in a field. The number -1 means that the field index is not valid. This action is only valid when using IBM 5250 or 5555 emulation.

Parameters

*Field Index*  The index of the field.

Format

Get_Num_Field_Data_IDs (Field Index)

Example

```
Script(Get_Field_Data_ID_Te
String(strDataID)
Number(numDataIDs)
Number(counter)
Activate(From_Menu)
    numDataIDs = Get_Num_Field_Data_IDs(0)
    counter = 0
    While(Number_Less_Than(counter, numDataIDs))
        strDataID = Get_Field_Data_ID(0, counter)
        Ask_OK(strDataID, "Data ID for Field 0")
        counter = Number_Plus(counter, 1)
    End_While
Return
```

See Also

Get_Num.Fields, Get_Field_Index_Row_Text, Get_Field_Index_Column_Text, Get_Field_Row, Get_Field_length, Get_Field_Symbology_OPERATOR, Set_Field_Append_Scan_Data, Set_Field_Com_Data_Field, Set_Field_Prefix_Scan_Data, Get_Field_Append_Scan_Data, Get_Field_Column, Get_Num_Field_Symbology_IDs, Get_Field_Com_Data_Field, Set_Field_Data_ID, Set_Field_Symbology_ID, Set_Field_Symbology_ID
Get_Num_Field_Symbology_IDs

Get the number of symbology IDs in a field. The number -1 means the field index is not valid. This action is only valid when using IBM 5250 or 5555 emulation.

**Parameters**

Field Index  The index of the field.

**Format**

Get_Num_Field_Symbology_IDs (Field Index)

**Example**

Script(Get_Field_Symbology_ID_Test)
String(strSymbologyID)
Boolean(ok)
Number(numSymbologies)
Number(counter)
Activate(From_Menu)
Comment: Set some symbologies for field 0, then display them.
ok = Set_Field_Symbology_ID(0, "UPCE0", FALSE)
ok = Set_Field_Symbology_ID(0, "CODE 39", FALSE)
ok = Set_Field_Symbology_ID(0, "EAN8", FALSE)
umSymbologies = Get_Num_Field_Symbology_IDs(0)
counter = 0
While(Number_Less_Than(counter, numSymbologies))
   strSymbologyID = Get_Field_Symbology_ID(0, counter)
   Ask_OK(strSymbologyID, "Symbology for Field 0")
   counter = Number_Plus(counter, 1)
End_While
Return

**See Also**

Get_Num_Fields, Get_Field_Index_Row_Text, Get_Field_Index_Column_Text, Get_Field_Row, Get_Field_Length, Get_Field_Symbology.Operator, Set_Field_Append_Scan_Data, Set_Field_Com_Data_Field, Set_Field_Prefix_Scan_Data, Get_Field_Append_Scan_Data, Get_Field_Column, Get_Num_Field_Data_IDs, Get_Field_Com_Data_Field, Set_Field_Data_ID, Set_Field_Symbology_ID, Get_Field_Symbology_ID
Get_Num_Fields

Get the number of fields on the screen. This action is only valid when using IBM 5250 or 5555 emulation.

Example

Script(Get_Field_Row_Column_Length)
String(strMessage)
Number(numFields)
Number(nLoops)
Number(nFieldRow)
Number(nFieldColumn)
Number(nFieldLength)
Activate(From_Menu)
  numFields = Get_Num_Fields
  Message(String_Combine("Number of fields:", Number_To_String_
Dec(Dec(numFields))), 60)
  nLoops = 0
  While(Number_Less_Than(nLoops, numFields))
    nFieldRow = Get_Field_Row(nLoops)
    nFieldColumn = Get_Field_Column(nLoops)
    nFieldLength = Get_Field_Length(nLoops)
    strMessage = String_Combine("Field:", Number_To_String_
Dec(nLoops))
    strMessage = String_Combine(strMessage, ":row")
    strMessage = String_Combine(strMessage, Number_To_String_
Dec(nFieldRow))
    strMessage = String_Combine(strMessage, ", column")
    strMessage = String_Combine(strMessage, Number_To_String_
Dec(nFieldColumn))
    strMessage = String_Combine(strMessage, ", length")
    strMessage = String_Combine(strMessage, Number_To_String_
Dec(nFieldLength))
    Ask_OK(strMessage, "Field Info")
    nLoops = Number_Plus(nLoops, 1)
  End_While
  Comment: The following should return zero because the field index is invalid.
  nFieldRow = Get_Field_Row(nLoops)
  nFieldColumn = Get_Field_Column(nLoops)
  nFieldLength = Get_Field_Length(nLoops)
Return
See Also

Get_Field_Index_Row_Text, Get_Field_Index_Column_Text, Get_Field_Row, Get_Field_Length, Get_Field_Symbology_Operator, Set_Field_Append_Scan_Data, Set_Field_Com_Data_Field, Set_Field_Prefix_Scan_Data, Get_Field_Append_Scan_Data, Get_Field_Column, Get_Num_Field_Data_IDS, Get_Num_Field_Symbology_IDS, Get_Field_Com_Data_Field, Set_Field_Data_ID, Set_Field_Symbology_ID, Get_Field_Symbology_ID
Get_Position_Column

Get the column number on which the cursor is currently located. The left-most column is 1.

Example

```plaintext
Script(Screen_Info)
String(StrMessage)
Number(nColumns)
Number(nRows)
Number(nPositionRow)
Number(nPositionColumn)
Activate(From_Menu)
    nRows = Get_Screen_Rows
    nColumns = Get_Screen_Columns
    nPositionRow = Get_Position_Row
    nPositionColumn = Get_Position_Column
    strMessage = String_Combine("Screen:", Number_To_String_Decimal(nRows))
    strMessage = String_Combine(strMessage, "rows,")
    strMessage = String_Combine(strMessage, Number_To_String_Decimal(nColumns))
    strMessage = String_Combine(strMessage, "columns")
    Message(strMessage, 10)
    strMessage = String_Combine("Cursor position: row", Number_To_String_Decimal(nPositionRow))
    strMessage = String_Combine(strMessage, " column")
    strMessage = String_Combine(strMessage, Number_To_String_Decimal(nPositionColumn))
    Ask_OK(strMessage, "Screen_Info")
Return
```

See Also

Get_Screen_Columns, Get_Screen_Rows, Get_Position_Row, Set_Cursor_Position, Get_Field_Row, Get_Field_Column
Get_Position_Row

Get the row number on which the cursor is currently located. The top-most row is 1.

Example

```plaintext
Script(Screen_Info)
String(StrMessage)
Number(nColumns)
Number(nRows)
Number(nPositionRow)
Number(nPositionColumn)
Activate(From_Menu)
   nRows = Get_Screen_Rows
   nColumns = Get_Screen_Columns
   nPositionRow = Get_Position_Row
   nPositionColumn = Get_Position_Column
   strMessage = String_Combine("Screen:", Number_To_String.Decimal(nRows))
   strMessage = String_Combine(strMessage, "rows,")
   strMessage = String_Combine(strMessage, Number_To_String.
   Decimal(nColumns))
   strMessage = String_Combine(strMessage, "columns")
   Message(strMessage, 10)
   strMessage = String_Combine("Cursor position: row", Number_To_String.
   Decimal(nPositionRow))
   strMessage = String_Combine(strMessage, " column")
   strMessage = String_Combine(strMessage, Number_To_String.
   Decimal(nPositionColumn))
   Ask_OK(strMessage, "Screen_Info")
   Return
```

See Also

Get_Screen_Columns, Get_Screen_Rows, Get_Position_Column, Set_Cursor_Position, Get_ Field_Row, Get_Field_Column
Get_Scan_Type_Name

Get the name of the scan type. An empty string is returned if the scan type is not recognized. Scanner symbology values can be found in Symbologies and Values on page 369.

Parameters

Scan Type The scan type number.

Format

Get_Scan_Type_Name (Scan Type)

Return Value

Returns the name of the supplied scan type.

Example

Script( Get_Scan_Type_Name_Test )
String( barcode )
String( strScanType )
Number( type )
Activate( On_Input, barcode, type )
    strScanType = Get_Scan_Type_Name( type )
    Ask_OK( strScanType, "Scan Type Name" )
    type = Ask_Number( "Enter a scan type", "Get_Screen_Text_Test", 0, 255, 60 )
    strScanType = Get_Scan_Type_Name( type )
    Ask_OK( strScanType, "Scan Type Name" )
Return

See Also

Scan_String, Get_Scan_Type_Value
Get_Scan_Type_Value

Get the number value of the supplied scan type name. A value of 0 is returned if the scan type name is not recognized. Scanner symbology values can be found in Symbologies and Values on page 369.

Parameters

Scan Type Name  The name of the scan type.

Format

Get_Scan_Type_Value (Scan Type Name)

Return Value

Returns the value of the supplied scan type name.

Example

Script( Get_Scan_Type_Value_Test )
String( strScanType )
String( strMessage )
Number( nScanNumberValue )
Activate( From Menu )
   strScanType = Ask_String_Uppercase( "Enter the scan type, like 
   ""UPCA"", "Get_Scan_Type_Value", 1, 99, "" )
   nScanNumberValue = Get_Scan_Type_Value( strScanType )
   strMessage = String_Combine( "Scan value for ", strScanType )
   strMessage = String_Combine( strMessage, ": " )
   strMessage = String_Combine( strMessage, Number_To_String_Decimal( nScanNumberValue ) )
   Message( strMessage, 7 )
   Return

See Also

Get_Num_Fields, Get_Field_Index_Row_Text, Get_Field_Index_Column_Text, Get_Field_Row, Get_Field_Length, Get_Field_Symbology_Operator, Set_Field_Append_Scan_Data, Set_Field_Com_Data_Field, Set_Field_Prefix_Scan_Data, Get_Field_Append_Scan_Data, Get_Field_Column, Get_Num_Field_Data_IDs, Get_Num_Field_Symbology_IDs, Get_Field_Com_Data_Field, Set_Field_Data_ID, Set_Field_Symbology_ID, Get_Field_Symbology_ID
Chapter 7: Overview of Actions

Get_Screen_Columns

Gets the number of columns on the screen. This is the total number of columns, not the number of columns visible.

Example

Script(Screen_Info)
String(StrMessage)
Number(nColumns)
Number(nRows)
Number(nPositionRow)
Number(nPositionColumn)
Activate(From_Menu)

    nRows = Get_Screen_Rows
    nColumns = Get_Screen_Columns
    nPositionRow = Get_Position_Row
    nPositionColumn = Get_Position_Column
    strMessage = String.Combine("Screen:", Number.To_String.Decimal(nRows))
    strMessage = String.Combine(strMessage, "rows,"
    strMessage = String.Combine(strMessage, Number.To_String.Decimal(nColumns))
    strMessage = String.Combine(strMessage, "columns")
    Message(strMessage, 10)
    strMessage = String.Combine(String.Combine("column")
    strMessage = String.Combine(strMessage, Number.To_String.Decimal(nPositionColumn))

Ask_OK(strMessage, "Screen_Info")
Return

See Also

Get_Screen_Rows, Get_Position_Column, Get_Position_Row, Get_Time, Get_Field_Row, Get_Field_Column, Set_Cursor_Position
Get_Screen_Rows

Get the number of rows on the screen. This is the total number of rows, not the number of rows visible.

Example

```plaintext
Script(Screen_Info)
String(StrMessage)
Number(nColumns)
Number(nRows)
Number(nPositionRow)
Number(nPositionColumn)
Activate(From_Menu)
    nRows = Get_Screen_Rows
    nColumns = Get_Screen_Columns
    nPositionRow = Get_Position_Row
    nPositionColumn = Get_Position_Column
    strMessage = String_Combine("Screen: ", Number_To_String_Decimal(nRows))
    strMessage = String_Combine(strMessage, "rows,")
    strMessage = String_Combine(strMessage, Number_To_String_Decimal(nColumns))
    strMessage = String_Combine(strMessage, "columns")
    Message(strMessage, 10)
    strMessage = String_Combine("Cursor position: row", Number_To_String_Decimal(nPositionRow))
    strMessage = String_Combine(strMessage, " column")
    strMessage = String_Combine(strMessage, Number_To_String_Decimal(nPositionColumn))
    Ask_OK(strMessage, "Screen_Info")
Return
```

See Also

Get_Screen_Columns, Get_Position_Column, Get_Position_Row, Get_Field_Row, Get_Field_Column, Set_Cursor_Position
Get_Screen_Text

Get the text at the specified location. The top row is row 1; the left-most column is column 1.

Parameters

Row The row of text to return.
Column The column of text to return.

Format

Get_Screen_Text (Row, Column)

Return Value

Returns the text starting at the specified screen position up to the right side of the display.

Example

Script(Get_Screen_Text_Test)
String(strScreenText)
Number(nRow)
Number(nColumn)
Activate(From_Menu)
    nRow = Ask_Number("Enter row, top row is 1", "Get_Screen_Text_Test", 1, 99, 1)
    nColumn = Ask_Number("Enter column, left-most column is 1", "Get_Screen_Text_Test", 1, 99, 1)
    strScreenText = Get_Screen_Text(nRow, nColumn)
    Ask_OK(strScreenText, "Screen Text")
Return

See Also

Get_Screen_Text_Columns, Get_Position_Row, Get_Screen_Text_Length, Get_Field_Index, Get_Field_Row, String_Equal, Ask_String, Get_Screen_Rows, Search_Screen, Speech_To_Text
Get_Screen_Text_Columns

Get text from a row on the screen, limited by the specified number of columns. The string will not include information past the number of columns specified.

Parameters

Row          The row of text to return.
Column       The column of text to return.
Number of Columns The maximum number of columns to get from the screen.

Format

Get_Screen_Text_Columns (Row, Column, Number of Columns)

Return Value

Returns the text starting at the specified screen position up to the right side of the display.

Example

Script( Get_Screen_Text_Columns_Test )
String( strScreenText )
Number( nRow )
Number( nColumn )
Number( nMaxColumns )
Activate( From_Menu )
   nRow = Ask_Number( "Enter row, top row is 1", "Get_Screen_Text_Test", 1, 99, 1 )
   nColumn = Ask_Number( "Enter column, top column is 1", "Get_Screen_Text_Test", 1, 99, 1)
   nMaxColumns = Ask_Number( "Enter maximum number of columns", "Get_Screen_Text_Test", 1, 99, 10 )
   strScreenText = Get_Screen_Text_Columns(nRow, nColumn, nMaxColumns)
   Ask_OK( strScreenText, "Screen Text" )
Return

See Also

Get_Screen_Text, Get_Screen_Text_Length, Get_Field_Index, Get_Field_Row, Speech_To_Text, Search_Screen, Get_Screen_Rows, Get_Position_Row, Ask_String, String_Equal
Get_Screen_Text_Length

Get the specified amount of text on the screen. The string will be truncated if it is longer than the number of characters specified. The top row is row 1; the left-most column is 1.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row</td>
<td>The row of text to return.</td>
</tr>
<tr>
<td>Column</td>
<td>The column of text to return.</td>
</tr>
<tr>
<td>Maximum Length</td>
<td>The maximum number of characters to get from the screen.</td>
</tr>
</tbody>
</table>

Format

Get_Screen_Text_Length (Row, Column, Maximum Length)

Return Value

Returns the text starting at the specified screen position up to the right side of the display.

Example

```
Script(Get_Screen_Text_Length_Test)
String(strScreenText)
Number(nRow)
Number(nColumn)
Number(nMaxCharacters)
Activate(From_Menu)
    nRow = Ask_Number("Enter row, top row is 1", "Get_Screen_Text_Test", 1, 99, 1)
    nColumn = Ask_Number("Enter column, top column is 1", "Get_Screen_Text_Test", 1, 99, 1)
    nMaxCharacters = Ask_Number("Enter maximum text length", "Get_Screen_Text_Test", 1, 99, 25)
    strScreenText = Get_Screen_Text_Length(nRow, nColumn, nMaxCharacters)
    Ask_OK(strScreenText, "ScreenText")
Return
```

See Also

Speech_To_Text, Search_Screen, Get_Field_Index, Get_Field_Row, Get_Screen_Text, Get_Screen_Text_Columns, Get_Screen_Rows, Get_Position_Row, String_Equal, Ask_String
Get_Session_Number

Get the number for the session in which this script is executing.

Example

Script(Get_Session_Number_Test)
String(strMessage)
String(strSessionNumber)
Number(nSession)
Activate(From_Menu)
    nSession = Get_Session_Number
    strSessionNumber = Number_To_String_Decimal(nSession)
    strMessage = String_Combine("Session Number:", strSessionNumber)
Message(strMessage, 7)
Return

See Also

Get_MAC_Address, Get_IP_Address, Get_Workstation_ID
Get_Time

Get the amount of time passed since January 1, 2000.

Return Value

Returns the number of seconds that have elapsed since January 1, 2000.

Example

Script( Get_Time_Test )
Number( nStartTimeSeconds )
Number( nDelayMilliseconds )
Number( nDelaySeconds )
Number( nEndTimeSeconds )
Activate( From_Menu )
  nStartTimeSeconds = Get_Time
  nDelaySeconds = 2
  nDelayMilliseconds = Number_Multiply( nDelaySeconds, 1000 )
  Message( String_Combine( Number_To_String_Decimal( nDelaySeconds ), " second delay...", nDelaySeconds )
  Delay( 2000 )
  nEndTimeSeconds = Get_Time
  nDelaySeconds = Number_Minus( nEndTimeSeconds, nStartTimeSeconds )

  Message( String_Combine( "The delay, in seconds, was: ", Number_To_String_Decimal( nDelaySeconds ) ), 10 )
  Return

See Also

Delay, Get_Time_Since_Reset, Wait_For_Screen_Update_With_Timeout
Get_Time_Since_Reset

Gets the amount of time since the last reboot.

Return Value

Returns the number of milliseconds that the computer has been non-suspended since the last reboot.

Example

Script( Get_Time_Since_Reset_Test )
String( strTitle )
String( strMessage )
Number( nMilliseconds )
Number( nSeconds )
Number( nMinutes )
Number( nHours )
Number( nDays )
Activate( From_Menu )
   strTitle = "The time since the last reboot, excluding time the device was suspended"
   nMilliseconds = Get_Time_Since_Reset
   nSeconds = Number_Divide( nMilliseconds, 1000 )
   nMinutes = Number_Divide( nSeconds, 60 )
   nHours = Number_Divide( nMinutes, 60 )
   nDays = Number_Divide( nHours, 24 )
   strMessage = String_Combine( Number_To_String.Decimal( nMilliseconds ),
                              " milliseconds = " )
   strMessage = String_Combine( strMessage, Number_To_String.Decimal( nSeconds ) )
   strMessage = String_Combine( strMessage, " seconds = " )
   strMessage = String_Combine( strMessage, Number_To_String.Decimal( nMinutes ) )
   strMessage = String_Combine( strMessage, " minutes = " )
   strMessage = String_Combine( strMessage, Number_To_String.Decimal( nHours ) )
   strMessage = String_Combine( strMessage, " hours = " )
   strMessage = String_Combine( strMessage, Number_To_String.Decimal( nDays ) )
   strMessage = String_Combine( strMessage, " days" )
Ask_OK( strMessage, strTitle )
Return
See Also

Delay, Wait_For_Screen_Update-With_Timeout, Get_Time, Reboot
Get_Workstation_ID

Get the Workstation ID of the device. This is only valid when using IBM emulation (3270, 5250, or 5555) and a Workstation ID has been specified for the current host profile. Otherwise, an empty string is returned.

**Return Value**

Returns the current Workstation ID.

**Example**

```plaintext
Script(Get_Workstation_ID_Test)
String(strWorkstationID)
  strWorkstationID = Get_Workstation_ID
  Ask_OK(strWorkstationID, "Workstation ID")
Return
```

**See Also**

Get_MAC_Address, Get_IP_Address, Get_Session_Number
Goto

Jumps to the supplied label.

Parameters

Label  The label where the script starts running after the Goto.

Format

Goto: (Label)

Example

Script(Goto_Test)
  Activate(From_Menu)
  Goto: end_script
  Message("Skipped...", 0)
  Label: end_script
  Message("The End.", 0)
  Return

See Also

Label
If

If the test is TRUE, the script continues executing until the next Else or EndIf statement. Otherwise, only executes actions (if any) between the next Else and EndIf statements.

Parameters

- **Test**: If TRUE, then the next set of actions get executed, up to the Else or EndIf, whichever comes first.

Format

```
If (Test)
```

Example

```
Script( If_Test )
Boolean( bOK )
Activate( From_Menu )
    Message_Clear
    bOK = Ask_OK_Cancel( "Press OK to see another message.", "Press OK", FALSE )
    If( bOK )
        Message( "bOK is TRUE", 0 )
    End_If
End_If
```

See Also

- If_Not, Else, End_If
If_Not

If the test is FALSE, the script continues executing until the next `Else` or `EndIf` statement. Otherwise, only executes actions (if any) between the next `Else` and `EndIf` statements.

**Parameters**

*Test*  
If FALSE, then the next set of actions get executed, up to the `Else` or `End_If`, whichever comes first.

**Format**

`If_Not (Test)`

**Example**

```wolfram
Script( If_Not_Test )
Boolean( bOK )
Activate( From_Menu )
    Message_Clear
    bOK = Ask_OK_Cancel( "Press Cancel to see another message.", "Press Cancel", FALSE )
    If_Not( bOK )
        Message( "bOK is FALSE", 0 )
    End_If
End_If
Return```

**See Also**

*If, Else, End_If*
Keyboard_Disable

Disables the keyboards. The keyboards will remain disabled until a script calls this action with a FALSE Disable value.

If Disable is TRUE, the hardware and on-screen keyboards for the session are disabled or the session is disconnected.

Parameters

Value1  Boolean (Disable)

Format

Keyboard_Disable(Value1)

Example

Script( Keyboard_Disable_Test )
Activate( From_Menu )
    Keyboard_Disable( TRUE )
    Message( "Keyboard is disabled for a few seconds", 0 )
    Delay( 3000 )
    Keyboard_Disable( FALSE )
    Message( "Keyboard is enabled", 3 )
Return
Keypress_Capture

Begins a keypress capture. The Key Value is the Diagnostics Windows Keyboard Test value for the keypress (for example, 0079 would become 0x0079). When the specified key and modifier combination is pressed, the Boolean variable will be set to TRUE.

Parameters

Value1 The Key Value (integer).
Value2 The modifier (Shift, Ctrl, Alt, None).
Value3 A Boolean variable.

Format

Keypress_Capture(Value1, "Value2", Value3)

Example

Script( Ctrl-F10_Keypress_Test )
Boolean( bKeyPressed )
Boolean( bF1Pressed )
Activate( From_Menu )
Message( "F1 is captured; press control-F10 to exit", 0 )
Keypress_Capture( 0x79, "Ctrl", bKeyPressed )
Keypress_Capture( 0x70, "", bF1Pressed )
While_Not( bKeyPressed )
   Wait_For_Screen_Update
   If( bF1Pressed )
      bF1Pressed = FALSE
      Keypress_Capture_Stop( 0x70, "" )
      Message( "F1 Pressed; capturing stopped for F1.", 3 )
   End_If
End_While
Keypress_Capture_Stop( 0x79, "C" )
Keypress_Capture_Stop_All
Message( "Ctrl-F10 Pressed - script is done", 5 )
Return
Keypress_Capture_Stop

Stops capturing the specified key and modifier combination. Supported modifiers are Shift, Ctrl, Alt and None. An empty string is treated as None.

Parameters

Value1  The Key Value (integer).
Value2  The modifier (Shift, Ctrl, Alt, None).

Format

Keypress_Capture_Stop (Value1, "Value2")

Example

Script( Ctrl-F10_Keypress_Test )
Boolean( bKeyPressed )
Boolean( bF1Pressed )
Activate( From_Menu )
    Message( "F1 is captured; press control-F10 to exit", 0 )
    Keypress_Capture( 0x79, "Ctrl", bKeyPressed )
    Keypress_Capture( 0x70, "", bF1Pressed )
While_Not( bKeyPressed )
    Wait_For_Screen_Update
    If( bF1Pressed )
        bF1Pressed = FALSE
        Keypress_Capture_Stop( 0x70, "" )
        Message( "F1 Pressed; capturing stopped for F1.", 3 )
    End_If
End_While
Keypress_Capture_Stop( 0x79, "C" )
Keypress_Capture_Stop All
Message( "Ctrl-F10 Pressed - script is done", 5 )
Return
Keypress_Capture_Stop_All

Stops capturing all key and modifier combinations.

Example

Script( Ctrl-F10_Keypress_Test )
Boolean( bKeyPressed )
Boolean( bF1Pressed )
Activate( From_Menu )
  Message( "F1 is captured; press control-F10 to exit", 0 )
  Keypress_Capture( 0x79, "Ctrl", bKeyPressed )
  Keypress_Capture( 0x70, "", bF1Pressed )
  While_Not( bKeyPressed )
    Wait_For_Screen_Update
    If( bF1Pressed )
      bF1Pressed = FALSE
      Keypress_Capture_Stop( 0x70, "" )
      Message( "F1 Pressed; capturing stopped for F1.", 3 )
    End_If
  End_While
Keypress_Capture_Stop( 0x79, "C" )
Keypress_Capture_Stop_All
Message( "Ctrl-F10 Pressed - script is done", 5 )
Return
**Keypress_Key**

Sends a single keypress to the Terminal Emulation session. This is useful for emulation keys that Keypress_String cannot handle.

**Parameters**

| Emulation Key Value | The number value of the key to send to the Terminal Emulation session. |

**Format**

```
Keypress_Key ("Emulation Key Value")
```

**Example**

```
Script(Test_Keypress)
   Activate(From_Menu)
       Keypress_String("N")
       Keypress_Key("VT220", "Enter")
   Return
```

**See Also**

Keypress_String, Scan_String, Set_Cursor_Position
Keypress_String

Creates one or more key presses to send the supplied string to the Terminal Emulation session.

Parameters

Characters The text characters to send to the Terminal Emulation session.

Format

Keypress_String ("Characters")

Example

Script(Test_Keypress)
   Activate(From_Menu)
      Keypress_String("N")
      Keypress_Key("VT220", "Enter")
   Return

See Also

Keypress_Key, Scan_String, Set_Cursor_Position
Label

Label to which a Goto can jump.

Parameters

Label  Identifies a line in the script for using Goto to change where the script is running.

Format

Label: (Label)

Example

Script(Goto_Test)
  Activate(From_Menu)
  Goto: end_script
  Message("Skipped...", 0)
  Label: end_script
  Message("The End.", 0)
  Return

See Also

Goto
Logging_Off

Turns off logging for the script.

Example

```plaintext
Script( Logging_Off_Test )
Activate( From_Menu )
    Logging_On( "TestingLogfile.txt", FALSE )
    Message( "A short message, followed by a delay...", 3 )
    Delay( 1500 )
    Logging_Off
    Message( "This message does not get logged", 3 )
Return
```

See Also

Logging_On
Logging_On

Creates a log file that records all subsequent script execution activity. This can be useful while developing a script, but is not recommended for production use. Logging is only turned on for the current script. Scripts called by this script will not have logging enabled.

If Overwrite Previous is TRUE, a previous log file will be overwritten. Otherwise, the new information will be appended to the existing file.

Parameters

<table>
<thead>
<tr>
<th>File Path</th>
<th>The log file path name.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overwrite Previous</td>
<td>Indicates whether the previous log file is overwritten.</td>
</tr>
</tbody>
</table>

Format

Logging_On ("File Path", Overwrite Previous)

Example

Script(Logging_Test)
    Activate(From_Menu)
    Logging_On("TestingLogfile.txt", FALSE)
    Message("A short message, followed by a delay...", 3)
    Delay(1500)
    Logging_Off
    Return

See Also

Logging_Off, Get_MAC_Address, Get_IP_Address, Get_Workstation_ID, Get_Session_Number
**Message**

Displays a message on the Terminal Emulation screen. Use 0 for the message box to be displayed until the user selects **OK**.

If the time-out value is greater than 0, the message is removed after that number of seconds elapses.

**Parameters**

- **Message**
  
  The text that appears in the message box.

- **Timeout**
  
  After the designated number of seconds until the message is removed.

**Format**

```
Message ("Message", Timeout)
```

**Example**

```
Script(Message_Test)
  Activate(From_Menu)
  Message("Message shows for five seconds", 5)
  Return
```

**See Also**

- **Message_Clear**, **Ask_OK**
Message_Clear

Clears the message on the Terminal Emulation screen.

Example

```plaintext
Script(Message_Test)
  Activate(From_Menu)
  Message("Waiting for screen update", 0)
  Wait_For_Screen_Update
  Message_Clear
  Return
```

See Also

Message
**Number_Divide**

Divide the first term by the second term and return the product. Because the numbers are integers, the remainder is ignored. For example, 7 divided by 3 would return 2.

**Parameters**

- **Number1**  The first term.
- **Number2**  The second term.

**Format**

```plaintext
Number_Divide (Number1, Number2)
```

**Example**

```plaintext
Script(Number_Actions)
String(strMessage)
String(strTitle)
String(strNumbers)
Number(nNumber1)
Number(nNumber2)
Number(nSum)
Number(nDifference)
Number(nProduct)
Number(nQuotient)
Number(nRemainder)
Activate(From_Menu)
    nNumber1 = Ask_Number("Enter the first number", "Number_Actions", 0, 2147483647, 36)
    nNumber2 = Ask_Number("Enter the second number", "Number_Actions", 0, 2147483647, 5)
    nSum = Number_Plus(nNumber1, nNumber2)
    nDifference = Number_Minus(nNumber1, nNumber2)
    nProduct = Number_Multiply(nNumber1, nNumber2)
    nQuotient = Number_Divide(nNumber1, nNumber2)
    nRemainder = Number_Divide_Remainder(nNumber1, nNumber2)
    strNumbers = String_Combine(Number_To_String.Decimal(nNumber1), ",")
    strNumbers = String_Combine(strNumbers, Number_To_String.Decimal(nNumber2))
    strTitle = String_Combine("Number_Plus", strNumbers)
    Ask_OK(Number_To_String.Decimal(nSum), strTitle)
    strTitle = String_Combine("Number_Minus", strNumbers)
    Ask_OK(Number_To_String.Decimal(nDifference), strTitle)
```
strTitle = String_Combine("Number_Multiply", strNumbers)
Ask_OK(Number_To_String.Decimal(nProduct), strTitle)
strTitle = String_Combine("Number_Divide", strNumbers)
Ask_OK(Number_To_String.Decimal(nQuotient), strTitle)
strTitle = String_Combine("Number_Divide_Remainder", strNumbers)
Ask_OK(Number_To_String.Decimal(nRemainder), strTitle)
Return

See Also

Number_Set, Number_Plus, Number_Minus, Number_Multiply, Number_Divide_Remainder, Ask_Number, Number_Equal
Number_Divide_Remainder

Divide the first term by the second term and return the remainder. For example, 7 divided by 3 would return a remainder of 1.

Parameters

Number1  The first term.

Number2  The second term.

Format

Number_Divide_Remainder (Number1, Number2)

Example

Script(Number_Actions)
String(strMessage)
String(strTitle)
String(strNumbers)
Number(nNumber1)
Number(nNumber2)
Number(nSum)
Number(nDifference)
Number(nProduct)
Number(nQuotient)
Number(nRemainder)
Activate(From_Menu)
  nNumber1 = Ask_Number("Enter the first number", "Number_Actions", 0, 2147483647, 36)
  nNumber2 = Asl_Number("Enter the second number", "Number_Actions", 0, 2147483647, 5)
  nSum = Number_Plus(nNumber1, nNumber2)
  nDifference = Number_Minus(nNumber1, nNumber2)
  nProduct = Number_Multiply(nNumber1, nNumber2)
  nQuotient = Number_Divide(nNumber1, nNumber2)
  nRemainder = Number_Divide_Remainder(nNumber1, nNumber2)
  strNumbers = String_Combine(Number_To_String_Decimal(nNumber1), ",")
  strNumbers = String_Combine(strNumbers, Number_To_String_Decimal(nNumber2))
  strTitle = String_Combine("Number_Plus", strNumbers)
  Ask_OK(Number_To_String_Decimal(nSum), strTitle)
  strTitle = String_Combine("Number_Minus", strNumbers)
  Ask_OK(Number_To_String_Decimal(nDifference), strTitle)
strTitle = String_Combine("Number_Multiply", strNumbers)
Ask_OK(Number_To_String_Decimal(nProduct), strTitle)
strTitle = String_Combine("Number_Divide", strNumbers)
Ask_OK(Number_To_String_Decimal(nQuotient), strTitle)
strTitle = String_Combine("Number_Divide_Remainder", strNumbers)
Ask_OK(Number_To_String_Decimal(nRemainder), strTitle)
Return

See Also

Number_Set, Number_Plus, Number_Minus, Number_Multiply, Number_Divide, Ask_Number, Number_Equal
**Number_Equal**

Compare two numbers and determine whether they are equal.

**Parameters**

- **Test1**  Gets compared to Test2.
- **Test2**  Gets compared to Test1.

**Format**

Number_Equal (Test1, Test2)

**Return Value**

Returns a Boolean. TRUE if Test1 is the same as Test2, FALSE otherwise.

**Example**

Script(Number_Equal_Test)
Number(nEntered1)
Number(nEntered2)
Boolean(bEqual)
Activate(From_Menu)
   nEntered1 = Ask_Number("Type the first number", "Number_Equal_Test", 0, 999, 0)
   nEntered2 = Ask_Number("Type the second number", "Number_Equal_Test", 0, 999, 0)
   bEqual = Number_Equal(nEntered1, nEntered2)
   If(bEqual)
      Message("First number is equal to the second number", 5)
   Else
      Message("First number is not equal to the second number", 5)
   End_If
Return

**See Also**

- Number_Less_Than
- Number_Less_Than_Or_Equal
- Number_Greater_Than_Or_Equal
- Number_Greater_Than
- Number_Not_Equal
- Number_Set
- Ask_Number
- String_To_Number
- Decimal
**Number_Greater_Than**

Compare two numbers and determine if one is greater than the other.

**Parameters**

- **Test1** Gets compared to Test2.
- **Test2** Gets compared to Test1.

**Format**

```
Number_Greater_Than (Test1, Test2)
```

**Return Value**

Returns a Boolean. TRUE if Test1 is larger than Test2, FALSE otherwise.

**Example**

```plaintext
Script(Number_Greater_Than_Test)
Number(nEntered1)
Number(nEntered2)
Boolean(bGreaterThan)
Activate(From_Menu)
   nEntered1 = Ask_Number("Type the first number", "Number_Greater_Than_Test", 0, 999, 0)
   nEntered2 = Ask_Number("Type the second number", "Number_Greater_Than_Test", 0, 999, 0)
   bGreaterThan = Number_Greater_Than(nEntered1, nEntered2)
   If(bGreaterThan)
      Message("First number is greater than second number", 5)
   Else
      Message("First number is not greater than second number", 5)
   End_If
Return
```

**See Also**

- Number_Less_Than, Number_Less_Than_Or_Equal, Number_Equals, Number_Greater_Than_Or_Equal, Number_Not_Equals, Number_Set, Ask_Number, String_To_Number_Decimal
Number_Greater_Than_Or_Equal

Compare two numbers and determine if one is greater than the other or if they are equal.

Parameters

Test1  Gets compared to Test2.
Test2  Gets compared to Test1.

Format

Number_Greater_Than_Or_Equal (Test1, Test2)

Return Value

Returns a Boolean. TRUE if Test1 is no smaller than Test2, FALSE otherwise.

Example

Script(Number_Greater_Than_Or_Equal_Test)
Number(nEntered1)
Number(nEntered2)
Boolean(bGreaterThan)
Activate(From_Menu)
    nEntered1 = Ask_Number("Type the first number", "Number_Greater_Than_Or_Equal_Test", 0, 999, 0)
    nEntered2 = Ask_Number("Type the second number", "Number_Greater_Than_Or_Equal_Test", 0, 999, 0)
    bGreaterThan = Number_Greater_Than_Or_Equal(nEntered1, nEntered2)
    If(bGreaterThan)
        Message("First number is greater than or equal to the second number", 5)
    Else
        Message("First number is not greater than or equal to the second number", 5)
    End_If
Return

See Also

Number_Less_Than, Number_Less_Than_Or_Equal, Number_Equal, Number_Greater_Than, Number_Not_Equal, Number_Set, Ask_Number, String_To_Number_Decimal
Number_Less_Than

Compares two numbers and returns TRUE if Test1 is smaller than Test2, FALSE otherwise. The largest number is 2147483647. The smallest number is -2147483648

Parameters

- **Test1** Gets compared to Test2.
- **Test2** Gets compared to Test1.

Format

Number_Less_Than (Test1, Test2)

Return Value

Returns a Boolean. TRUE if Test1 is smaller than Test2, FALSE otherwise.

Example

```plaintext
Script(Number_Less_Than_Test)
Boolean(bLessThan)
Number(nEntered1)
Number(nEntered2)
Activate(From_Menu)
    nEntered1 = Ask_Number("Type the first number", "Number_Less_Than_Test", 0, 2147483647, 0)
    nEntered2 = Ask_Number("Type the second number", "Number_Less_Than_Test", 0, 999, 0)
    bLessThan = Number_Less_Than(nEntered1, nEntered2)
    If(bLessThan)
        Message("First number is less than second number", 5)
    Else
        Message("First number is not less than second number", 5)
End_If
Return
```

See Also

- Number_Less_Than_Or_Equal, Number_Equal, Number_Greater_Than_Or_Equal, Number_Greater_Than, Number_Not_Equal, Number_Set, Ask_Number, String_To_Number_Decimal
Chapter 7: Overview of Actions

Number_Less_Than_Or_Equal

Compare two numbers and determine if one is less than the other or if they are equal.

Parameters

- Test1 Gets compared to Test2.
- Test2 Gets compared to Test1.

Format

Number_Less_Than_Or_Equal (Test1, Test2)

Return Value

Returns a Boolean. TRUE if Test1 is no greater than Test2, FALSE otherwise.

Example

Script(Number_Less_Than_Or_Equal_Test)
Number(nEntered1)
Number(nEntered2)
Boolean(bLessThan)
Activate(From_Menu)
  nEntered1 = Ask_Number("Type the first number", "Number_Less_Than_Or_Equal_Test", 0, 999, 0)
  nEntered2 = Ask_Number("Type the second number", "Number_Less_Than_Or_Equal_Test", 0, 999, 0)
  bLessThan = Number_Less_Than_Or_Equal(nEntered1, nEntered2)
  If(bLessThan)
    Message("First number is less than or equal to the second number", 5)
  Else
    Message("First number is not less than or equal to the second number", 5)
  End_If
Return

See Also

Number_Less_Than, Number_Equal, Number_Greater_Than_Or_Equal, Number_Greater_Than, Number_Not_Equal, Number_Set, Ask_Number, String_To_Number_Decimal
Number_Minus

Subtract the second term from the first term to get the difference.

Parameters

- **Number1**  The first term.
- **Number2**  The second term.

Format

\[
\text{Number\_Minus}(\text{Number1, Number2})
\]

Return Value

Returns the value when Number2 is subtracted from Number1.

Example

```plaintext
Script(Number\_Actions)
String(strMessage)
String(strTitle)
String(strNumbers)
Number(nNumber1)
Number(nNumber2)
Number(nSum)
Number(nDifference)
Number(nProduct)
Number(nQuotient)
Number(nRemainder)
Activate(From\_Menu)
  nNumber1 = Ask\_Number("Enter the first number", "Number\_Actions", 0, 2147483647, 36)
  nNumber2 = Asl\_Number("Enter the second number", "Number\_Actions", 0, 2147483647, 5)
  nSum = Number\_Plus(nNumber1, nNumber2)
  nDifference = Number\_Minus(nNumber1, nNumber2)
  nProduct = Number\_Multiply(nNumber, nNumber2)
  nQuotient = Number\_Divide(nNumber1, nNumber2)
  nRemainder = Number\_Divide\_Remainder(nNumber1, nNumber2)
  strNumbers = String\_Combine(Number\_To\_String\_Decimal(nNumber1), ",", ")
  strNumbers = String\_Combine(strNumbers, Number\_To\_String\_Decimal(nNumber2))
  strTitle = String\_Combine("Number\_Plus", strNumbers)
```
Ask_OK(Number_To_String.Decimal(nSum), strTitle)
strTitle = String_Combine("Number_Minus", strNumbers)
Ask_OK(Number_To_String.Decimal(nDifference), strTitle)
strTitle = String_Combine("Number_Multiply", strNumbers)
Ask_OK(Number_To_String.Decimal(nProduct), strTitle)
strTitle = String_Combine("Number_Divide", strNumbers)
Ask_OK(Number_To_String.Decimal(nQuotient), strTitle)
strTitle = String_Combine("Number_Divide_Remainder", strNumbers)
Ask_OK(Number_To_String.Decimal(nRemainder), strTitle)
Return

See Also

Number_Set, Number_Plus, Number_Multiply, Number_Divide, Number_Divide_Remainder, Ask_Number, Number_Equal
**Number_Multiply**

Multiply the first term by the second term and returns the product. Each parameter may be a constant, variable, or action.

**Parameters**

- **Number1**  The first term.
- **Number2**  The second term.

**Format**

```
Number_Multiply (Number1, Number2)
```

**Example**

```
Script(Number_Actions)
String(strMessage)
String(strTitle)
String(strNumbers)
Number(nNumber1)
Number(nNumber2)
Number(nSum)
Number(nDifference)
Number(nProduct)
Number(nQuotient)
Number(nRemainder)
Activate(From_Menu)
    nNumber1 = Ask_Number("Enter the first number", "Number_Actions", 0, 2147483647, 36)
    nNumber2 = Asl_Number("Enter the second number", "Number_Actions", 0, 2147483647, 5)
    nSum = Number_Plus(nNumber1, nNumber2)
    nDifference = Number_Minus(nNumber1, nNumber2)
    nProduct = Number_Multiply(nNumber1, nNumber2)
    nQuotient = Number_Divide(nNumber1, nNumber2)
    nRemainder = Number_Divide_Remainder(nNumber1, nNumber2)
    strNumbers = String_Combine(Number_To_String_Decimal(nNumber1), ",")
    strNumbers = String_Combine(strNumbers, Number_To_String_Decimal(nNumber2))
    strTitle = String_Combine("Number_Plus", strNumbers)
    Ask_OK(Number_To_String_Decimal(nSum), strTitle)
    strTitle = String_Combine("Number_Minus", strNumbers)
    Ask_OK(Number_To_String_Decimal(nDifference), strTitle)
```
strTitle = String_Combine("Number_Multiply", strNumbers)
Ask_OK(Number_To_String_Decimal(nProduct), strTitle)
strTitle = String_Combine("Number_Divide", strNumbers)
Ask_OK(Number_To_String_Decimal(nQuotient), strTitle)
strTitle = String_Combine("Number_Divide_Remainder", strNumbers)
Ask_OK(Number_To_String_Decimal(nRemainder), strTitle)
Return

See Also

Number_Set, Number_Plus, Number_Minus, Number_Divide, Number_Divide_Remainder, Ask_Number, Number_Equal
Number_Not_Equal

Compares two numbers.

Parameters

Parameter1(Number):"Test1"  Gets compared to Test2

Parameter1(Number):"Test2"  Gets compared to Test1.

Format

Number_Not_Equal:"Test1"

Return Value

Returns a Boolean. FALSE if Test1 is the same as Test2, TRUE otherwise.

Example

Script( Number_Not_Equal_Test )
Boolean( bNotEqual )
Number( nEntered1 )
Number( nEntered2 )
Activate( From_Menu )
    nEntered1 = Ask_Number( "Type the first number", "Number_Not_Equal_Test", 0, 999, 15 )
    nEntered2 = Ask_Number( "Type the second number", "Number_Not_Equal_Test", 0, 999, 704 )
    bNotEqual = Number_Not_Equal( nEntered1, nEntered2 )
If( bNotEqual )
    Message( "First number is not equal to the second number", 8 )
Else
    Message( "First number is equal to the second number", 8 )
End_If

See Also

Number_Less_Than, Number_Less_Than_Or_Equal, Number_Equal, Number_Greater_Than_Or_Equal, Number_Greater_Than, Number_Set, Ask_Number, String_To_Number_Decimal
Number_Plus

Add two numbers together and return the sum. Each parameter may be a constant or a variable or an action.

Parameters

Number1  The first term.
Number2  The second term.

Format

Number_Plus (Number1, Number2)

Example

Script(Number_Actions)
String(strMessage)
String(strTitle)
String(strNumbers)
Number(nNumber1)
Number(nNumber2)
Number(nSum)
Number(nDifference)
Number(nProduct)
Number(nQuotient)
Number(nRemainder)
Activate(From_Menu)
  nNumber1 = Ask_Number("Enter the first number", "Number_Actions", 0,
  2147483647, 36)
  nNumber2 = Ask_Number("Enter the second number", "Number_Actions", 0,
  2147483647, 5)
  nSum = Number_Plus(nNumber1, nNumber2)
  nDifference = Number_Minus(nNumber1, nNumber2)
  nProduct = Number_Multiply(nNumber1, nNumber2)
  nQuotient = Number_Divide(nNumber1, nNumber2)
  nRemainder = Number_Divide_Remainder(nNumber1, nNumber2)
  strNumbers = String.Combine(Number_To_String.Decimal(nNumber1), ",")
  strNumbers = String.Combine(strNumbers, Number_To_String.
  Decimal(nNumber2))
  strTitle = String.Combine("Number_Plus", strNumbers)
  Ask_OK(Number_To_String.Decimal(nSum), strTitle)
  strTitle = String.Combine("Number_Minus", strNumbers)
  Ask_OK(Number_To_String.Decimal(nDifference), strTitle)
strTitle = String_Combine("Number_Multiply", strNumbers)
Ask_OK(Number_To_String_Decimal(nProduct), strTitle)
strTitle = String_Combine("Number_Divide", strNumbers)
Ask_OK(Number_To_String_Decimal(nQuotient), strTitle)
strTitle = String_Combine("Number_Divide_Remainder", strNumbers)
Ask_OK(Number_To_String_Decimal(nRemainder), strTitle)
Return

See Also

Number_Set, Number_Minus, Number_Multiply, Number_Divide, Number_Divide_Remainder, Ask_Number, Number_Equal
Number_Set

Set the value of a number variable. The equal sign (=) is the symbol for Number_Set in the Script Editor.

Parameters

Number  A constant, variable, or action.

Format

Number_Set (Number)

Return Value

Returns the value of the number.
Chapter 7: Overview of Actions

Example

```plaintext
Script(Number_Actions)
String(strMessage)
String(strTitle)
String(strNumbers)
Number(nNumber1)
Number(nNumber2)
Number(nSum)
Number(nDifference)
Number(nProduct)
Number(nQuotient)
Number(nRemainder)
Activate(From_Menu)
    nNumber1 = Ask_Number("Enter the first number", "Number_Actions", 0, 2147483647, 36)
    nNumber2 = Ask_Number("Enter the second number", "Number_Actions", 0, 2147483647, 5)
    nSum = Number_Plus(nNumber1, nNumber2)
    nDifference = Number_Minus(nNumber1, nNumber2)
    nProduct = Number_Multiply(nNumber1, nNumber2)
    nQuotient = Number_Divide(nNumber1, nNumber2)
    nRemainder = Number_Divide_Remainder(nNumber1, nNumber2)
    strNumbers = String_Combine(Number_To_String_Decimal(nNumber1), ",", )
    strNumbers = String_Combine(strNumbers, Number_To_String_Decimal(nNumber2))
    strTitle = String_Combine("Number_Plus", strNumbers)
    Ask_OK(Number_To_String_Decimal(nSum), strTitle)
    strTitle = String_Combine("Number_Minus", strNumbers)
    Ask_OK(Number_To_String_Decimal(nDifference), strTitle)
    strTitle = String_Combine("Number_Multiply", strNumbers)
    Ask_OK(Number_To_String_Decimal(nProduct), strTitle)
    strTitle = String_Combine("Number_Divide", strNumbers)
    Ask_OK(Number_To_String_Decimal(nQuotient), strTitle)
    strTitle = String_Combine("Number_Divide_Remainder", strNumbers)
    Ask_OK(Number_To_String_Decimal(nRemainder), strTitle)
Return
```

See Also

Number_Plus, Number_Minus, Number_Multiply, Number_Divide, Number_Divide_Remainder, Ask_Number, Number_Equal
**Number_To_Character**

Converts the specified number to the character value. For example, a number value of 87 would return a string consisting of a "W", the ASCII character value for 87.

**Parameters**

*Number*  The number to convert to a character.

**Format**

*Number_To_Character* (Number)

**Return Value**

Returns a string one character in length, where the value for that character is the supplied number.

**Example**

```wscript
Script( Number_To_Character_Test )
String( strCharacter )
String( strTitle )
Number( nToConvert )
Activate( From_Menu )

nToConvert = Ask_Number( "Enter a number that will be converted into a character", "Number_To_Character", 30, 126, 65 )
strCharacter = Number_To_Character( nToConvert )
strTitle = Number_To_String_Decimal( nToConvert )
strTitle = String_Combine( "Number_To_Character of ", strTitle )
Ask_OK( strCharacter, strTitle )
Return
```
Number_To_String_Binary

Gets the binary representation of the specified number.

Parameters

\[ \text{Number} \quad \text{The number to convert to binary.} \]

Format

\[ \text{Number_To_String_Binary (Number)} \]

Return Value

Returns a string with the binary representation of the number.

Example

\[
\begin{align*}
\text{Script(Number_Convert)} \\
\text{String(strEntered)} \\
\text{String(strBinary)} \\
\text{String(strHexLower)} \\
\text{String(strHexUpper)} \\
\text{String(strOctal)} \\
\text{Number(numEntered)} \\
\text{Activate(From_Menu)} \\
\quad \text{numEntered} = \text{Ask_Number("Enter the decimal number to convert", "Number_Convert", -22, 2000000000, 31)} \\
\quad \text{strEntered} = \text{Number_To_String_Decimal(numEntered)} \\
\quad \text{strBinary} = \text{Number_To_String_Binary(numEntered)} \\
\quad \text{strHexLower} = \text{Number_To_String_Hexadecimal_Lowercase(numEntered)} \\
\quad \text{strHexUpper} = \text{Number_To_String_Hexadecimal_Uppercase(numEntered)} \\
\quad \text{strOctal} = \text{Number_To_String_Octal(numEntered)} \\
\quad \text{Ask_OK(strBinary, String_Combine("Binary value of", strEntered))} \\
\quad \text{Ask_OK(strHexLower, String_Combine("Hex (lower case) value of ", strEntered))} \\
\quad \text{Ask_OK(strHexUpper, String_Combine("Hex (upper case) value of ", strEntered))} \\
\quad \text{Ask_OK(strOctal, String_Combine("Octal value of", strEntered))} \\
\quad \text{Return}
\end{align*}
\]
See Also

String_Set, String_Combine, String_Left, String_Right, String_Middle, String_Upper, String_Lower, String_Replace, String_Only_Characters, String_Strip_Characters, String_Trim_Spaces_Start, String_Trim_Spaces_End, Number_To[String_Octal, Number_To[String_Decimal, Number_To[String_Hexadecimal_Lowercase, Number_To[String_Hexadecimal_Uppercase, Ask_String, Ask/String_Password, Ask/String_Uppercase, Ask/String_Lowercase, String_Equal
**Number_To_String.Decimal**

Gets the decimal representation of the specified number.

**Parameters**

*Number*  The number to convert to decimal.

**Format**

Number_To_String.Decimal (Number)

**Return Value**

Returns a string with the decimal (base 10) representation of the number.

**Example**

Script(Number_Convert)
String(strEntered)
String(strBinary)
String(strHexLower)
String(strHexUpper)
String(strOctal)
Number(numEntered)
Activate(From_Menu)
  numEntered = Ask_Number("Enter the decimal number to convert", "Number_Convert", -22, 2000000000, 31)
  strEntered = Number_To_String.Decimal(numEntered)
  strBinary = Number_To_String.Binary(numEntered)
  strHexLower = Number_To_String_Hexadecimal_Lowercase(numEntered)
  strHexUpper = Number_To_String_Hexadecimal_Uppercase(numEntered)
  strOctal = Number_To_String_Octal(numEntered)
  Ask_OK(strBinary, String.Combine("Binary value of", strEntered))
  Ask_OK(strHexLower, String.Combine("Hex (lower case) value of ", strEntered))
  Ask_OK(strHexUpper, String.Combine("Hex (upper case) value of ", strEntered))
  Ask_OK(strOctal, String.Combine("Octal value of ", strEntered))
Return
See Also

String_Set, String_Combine, String_Left, String_Right, String_Middle, String_Upper, String_Lower, String_Replace, String_Only_Characters, String_Strip_Characters, String_Trim_Spaces_Start, String_Trim_Spaces_End, Number_To_String_Binary, Number_To_String_Octal, Number_To_String_Hexadecimal_Lowercase, Number_To_String_Hexadecimal_Uppercase, Ask_String, Ask_String_Password, Ask_String_Uppercase, Ask_String_Lowercase, String_Equal
**Number_To_String_Hexadecimal_Lowercase**

Gets the hexadecimal representation of the specified number.

**Parameters**

- **Number**  The number to convert to hexadecimal.

**Format**

```
Number_To_String_Hexadecimal_LowerCase (Number)
```

**Return Value**

Returns a string with the hexadecimal (base 16) representation of the number using lowercase characters.

**Example**

```script
Script(Number_Convert)
String(strEntered)
String(strBinary)
String(strHexLower)
String(strHexUpper)
String(strOctal)
Number(numEntered)
Activate(From_Menu)
  numEntered = Ask_Number("Enter the decimal number to convert", "Number_Convert", -22, 2000000000, 31)
  strEntered = Number_To_String_Decimal(numEntered)
  strBinary = Number_To_String_Binary(numEntered)
  strHexLower = Number_To_String_Hexadecimal_Lowercase(numEntered)
  strHexUpper = Number_To_String_Hexadecimal_Uppercase(numEntered)
  strOctal = Number_To_String_Octal(numEntered)
  Ask_OK(strBinary, String_Combine("Binary value of ", strEntered))
  Ask_OK(strHexLower, String_Combine("Hex (lower case) value of ", strEntered))
  Ask_OK(strHexUpper, String_Combine("Hex (upper case) value of ", strEntered))
  Ask_OK(strOctal, String_Combine("Octal value of ", strEntered))
Return
```
See Also

String_Set, String_Combine, String_Left, String_Right, String_Middle, String_Upper, String_Lower, String_Replace, String_Only_Characters, String_Strip_Characters, String_Trim_Spaces_Start, String_Trim_Spaces_End, Number_To_String_Binary, Number_To_String_Octal, Number_To_String.Decimal, Number_To_String.Hexadecimal_Uppercase, Ask_String, Ask_String_Password, Ask_String_Uppercase, Ask_String_Lowercase, String_Equal
Chapter 7: Overview of Actions

Number_To_String_Hexadecimal_Uppercase

Gets the hexadecimal representation of the specified number.

Parameters

Number  The number to convert to hexadecimal.

Format

Number_To_String_Hexadecimal_Uppercase (Number)

Return Value

Returns a string with the hexadecimal (base 16) representation of the number using uppercase characters.

Example

Script(Number_Convert)
String(strEntered)
String(strBinary)
String(strHexLower)
String(strHexUpper)
String(strOctal)
Number(numEntered)
Activate(From_Menu)
    numEntered = Ask_Number("Enter the decimal number to convert", "Number_Convert", -22, 2000000000, 31)
    strEntered = Number_To_String_Decimal(numEntered)
    strBinary = Number_To_String_Binary(numEntered)
    strHexLower = Number_To_String_Hexadecimal_Lowercase(numEntered)
    strHexUpper = Number_To_String_Hexadecimal_Uppercase(numEntered)
    strOctal = Number_To_String_Octal(numEntered)
    Ask_OK(strBinary, String_Combine("Binary value of", strEntered))
    Ask_OK(strHexLower, String_Combine("Hex (lower case) value of ", strEntered))
    Ask_OK(strHexUpper, String_Combine("Hex (upper case) value of ", strEntered))
    Ask_OK(strOctal, String_Combine("Octal value of ", strEntered))
Return
See Also

String_Set, String_Combine, String_Left, String_Right, String_Middle, String_Upper, String_Lower, String_Replace, String_Only_Characters, String_Strip_Characters, String_Trim_Spaces_Start, String_Trim_Spaces_End, Number_To_String_Binary, Number_To_String_Octal, Number_To_String_Decimal, Number_To_String_Hexadecimal_Lowercase, Ask_String, Ask_String_Password, Ask_String_Uppercase, Ask_String_Lowercase, String_Equal
**Number_To_String_Octal**

Gets the octal representation of the specified number.

**Parameters**

*Number*  The number to convert to octal.

**Format**

Number_To_String_Octal (Number)

**Return Value**

Returns a string with the octal (base 8) representation of the number.

**Example**

```
Script(Number_Convert)
String(strEntered)
String(strBinary)
String(strHexLower)
String(strHexUpper)
String(strOctal)
Number(numEntered)
Activate(From_Menu)
    numEntered = Ask_Number("Enter the decimal number to convert", "Number_Convert", -22, 2000000000, 31)
    strEntered = Number_To_String.Decimal(numEntered)
    strBinary = Number_To_String.Binary(numEntered)
    strHexLower = Number_To_String.Hexadecimal_Lowercase(numEntered)
    strHexUpper = Number_To_String.Hexadecimal_Uppercase(numEntered)
    strOctal = Number_To_String.Octal(numEntered)
    Ask_OK(strBinary, String.Combine("Binary value of", strEntered))
    Ask_OK(strHexLower, String.Combine("Hex (lower case) value of ", strEntered))
    Ask_OK(strHexUpper, String.Combine("Hex (upper case) value of ", strEntered))
    Ask_OK(strOctal, String.Combine("Octal value of ", strEntered))
Return
```
See Also

String_Set, String_Combine, String_Left, String_Right, String_Middle, String_Upper, String_Lower, String_Replace, String_Only_Characters, String_Strip_Characters, String_Trim_Spaces_Start, String_Trim_Spaces_End, Number_To_String_Binary, Number_To_String.Decimal, Number_To_String_Hexadecimal_Lowercase, Number_To_String_Hexadecimal_Uppercase, Ask_String, Ask_String_Password, Ask_String_Uppercase, Ask_String_Lowercase, String_Equal
**Play_Sound**

Causes the device to play the sound specified by the sound name. The sound name may be any `.wav` file located in the folder specified by the emulation parameters: **Emulation > Sound > Sound Resource** folder. The sound name may also be any of the sounds in the Resource Editor.

**Parameters**

- **Sound Name**  The name of the `.wav` file.

**Format**

```
Play_Sound ("Sound Name")
```

**Example**

```
Script(Play_Sound_Test)
  Activate(From_Menu)
  Comment: Play the Chimes sound from the Resource Editor.
  Play_Sound("Chimes")
  Return
```

**See Also**

- **Beep**, **Speech_From_Text**
Printer_Cancel

Instructs the printer to discard all Printer_Data information already received.

Example

Script( Printing_Example )
String( sScanData )
Number( sScanType )
Activate( On_Input, sScanData, sScanType )
  Comment: Prints a label containing displaying the barcode scanned.
  If_Not( Printer_Data( "!: 0 200 200 148 1\0D\0A", FALSE ) )
    Goto: Print_Error
  End_If
  If_Not( Printer_Data( "LABEL\0D\0A", FALSE ) )
    Goto: Print_Error
  End_If
  If_Not( Printer_Data( "BARCODE CODE39 2 1 45 30 70 ", FALSE ) )
    Goto: Print_Error
  End_If
  If( Number_Greater_Than( String_Length( sScanData ), 20 ) )
    Printer_Cancel
    Message( "Barcode is too long.", 5 )
    Return
  End_If
  If_Not( Printer_Data( sScanData, FALSE ) )
    Goto: Print_Error
  End_If
  If_Not( Printer_Data( "\0D\0A", TRUE ) )
    Goto: Print_Error
  End_If
  Comment: Offer to let the user print additional labels.
  While( Ask_Yes_No( "Do you want to print the label again?", "Reprint", TRUE ) )
    If_Not( Printer_Repeat )
      Goto: Print_Error
    End_If
  End_While
  Message( "Printing successful.", 5 )
  Return
Label: Print_Error
  Printer_Cancel
  Message( "Unable to print the label.", 5 )
  Return
Chapter 7: Overview of Actions

Printer_Data

Sends data directly to the currently defined printer. If All Data Sent is TRUE, then printing will begin after the latest data is sent. Otherwise, printing will wait for additional data.

Parameters

Value1  Print data (a string).
Value2  All Data Sent (Boolean).

Format

Print_Data("Value1"), Value2

Example

Script( Printing_Example )
String( sScanData )
Number( sScanType )
Activate( On_Input, sScanData, sScanType )
Comment: Prints a label containing displaying the barcode scanned.
If_Not( Printer_Data( "! 0 200 200 148 1\0D\0A", FALSE ) )
   Goto: Print_Error
End_If
If_Not( Printer_Data( "LABEL\0D\0A", FALSE ) )
   Goto: Print_Error
End_If
If_Not( Printer_Data( "BARCODE CODE39 2 1 45 30 70 ", FALSE ) )
   Goto: Print_Error
End_If
If( Number_Greater_Than( String_Length( sScanData ), 20 ) )
   Printer_Cancel
   Message( "Barcode is too long.", 5 )
   Return
End_If
If_Not( Printer_Data( sScanData, FALSE ) )
   Goto: Print_Error
End_If
If_Not( Printer_Data( "\0D\0A", TRUE ) )
   Goto: Print_Error
End_If
Comment: Offer to let the user print additional labels.
While( Ask_Yes_No( "Do you want to print the label again?", "Reprint", TRUE ) )
    If_Not( Printer_Repeat )
        Goto: Print_Error
    End_If
End_While

Message( "Printing successful.", 5 )
Return

Label: Print_Error
    Printer_Cancel
    Message( "Unable to print the label.", 5 )
    Return
**Printer_Repeat**

Instructs the printer to reprint the last item printed. If TRUE, the printer will produce the last item printed.

**Example**

```
Script( Printing_Example )
String( sScanData )
Number( sScanType )
Activate( On_Input, sScanData, sScanType )

Comment: Prints a label containing displaying the barcode scanned.
If_Not( Printer_Data( "! 0 200 200 148 1\0D\0A", FALSE ) )
    Goto: Print_Error
End_If
If_Not( Printer_Data( "LABEL\0D\0A", FALSE ) )
    Goto: Print_Error
End_If
If_Not( Printer_Data( "BARCODE CODE39 2 1 45 30 70 ", FALSE ) )
    Goto: Print_Error
End_If
If( Number_Greater_Than( String_Length( sScanData ), 20 ) )
    Printer_Cancel
    Message( "Barcode is too long.", 5 )
    Return
End_If
If_Not( Printer_Data( sScanData, FALSE ) )
    Goto: Print_Error
End_If
If_Not( Printer_Data( "\0D\0A", TRUE ) )
    Goto: Print_Error
End_If

Comment: Offer to let the user print additional labels.
While( Ask_Yes_No( "Do you want to print the label again?", "Reprint", TRUE ) )
    If_Not( Printer_Repeat )
        Goto: Print_Error
    End_If
End_While

Message( "Printing successful.", 5 )
Return

Label: Print_Error
```
Printer_Cancel
Message( "Unable to print the label.", 5 )
Return
Reboot

Reboots the device. Any subsequent commands will not be executed unless the reboot fails. If Cold Boot is TRUE, the device will cold boot. Some applications and settings may be lost. If Cold Boot is FALSE, the device will not perform a cold boot.

**NOTE:** Cold boot is only supported by some mobile devices.

**Parameters**

*Cold Boot*  Indicates whether the device will cold boot.

**Format**

Reboot (Cold Boot)

**Example**

Script(Reboot_Test)
   Activate(From_Menu)
      Comment: Do a regular reboot of the device, not a cold boot.
      Reboot(FALSE)
   Return

**See Also**

*Ask_Yes_No, Disconnect, Exit_Application, Abort_All, Suspend, Get_Time_Since_Reset*
Return

Exits the script normally. If this script was started by another script, the calling script’s variables are updated and the calling script resumes.

Example

```wisp
Script(Return_Test)
Activate(From_Menu)
    Comment: This script doesn’t do anything
Return
```

See Also

Abort, Abort_All, Disconnect, Exit_Application
Run_Application

Starts an application with the flags (optional) or a known type file with no flags. Specifying the full path to the application or file is recommended.

Parameters

Value1  The path to the application (a string).
Value2  The flags to run the application (a string).
Value3  Boolean (Wait Until Exit).

Format

Run_Application ("String", "String", Boolean)

Returns

If the application fails to start, a value of -1 will be returned.

If the "Wait Until Exit" value is TRUE, the value returned will be the exit code for the application. Otherwise, a value of 0 will be returned.

Example

Script ( Run_Application_Test )
String ( strExit )
Number ( nExitCode )
Activate ( From_Menu )
    nExitCode = Run_Application ( "Notepad.exe", "", TRUE )
    strExit = Number_To_String_Decimal ( nExitCode )
    Ask_OK ( strExit, "Exit Code" )
Return
Scan_String

Treats the string as scanned data of the type specified. Scanner symbology values can be found in Symbologies and Values on page 369.

**Parameters**

- **Characters**: The string that will be treated as scan data.
- **Scan Type**: The scanner symbology of the scanned data.

**Format**

```
Scan_String (Characters, Scan Type)
```

**Example**

```
Script(Scan_String_Test)
    String(sScanData)
    Number(nScanType)
    Activate(On_Input, sScanData, nScanType)
        Comment: See if this is a special barcode to indicate a keypress.
        Comment: You can adjust the barcode type and test strings to suit your purposes.
        If(Number_Equal(nScanType, Get_Scan_Type_Value("CODE 128")))
            If(String_Equal(sScanData, "Ctrl-A", 0, TRUE))
                Keypress_Key("VT220", "Ctrl-A")
                Return
            End_If
        End_If
    End_Forc
Comment: We didn’t use the scan data. Pass it along for standard processing.
    Scan_String(sScanData, nScanType)
    Return
```

**See Also**

- Keypress_String, Keypress_Key, Set_Cursor_Position, Get_Scan_Type_Name, Get_Scan_Type_Value
Chapter 7: Overview of Actions

Search_Screen

Searches the screen for the supplied text. The rows to be searched can be specified, where 1 is the top row. If the bottom row value is less than 1, searching continues to the bottom of the screen. If Ignore Case is TRUE, then upper-case and lower-case letters are considered to be equal.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search String</td>
<td>The string to find on the screen.</td>
</tr>
<tr>
<td>Top Search Row</td>
<td>The row in which the search begins.</td>
</tr>
<tr>
<td>Bottom Search Row</td>
<td>The row in which the search ends.</td>
</tr>
<tr>
<td>Ignore Case</td>
<td>Indicates whether the case of the letters is taken into consideration.</td>
</tr>
</tbody>
</table>

Format

Search_Screen (Search String, Top Search Row, Bottom Search Row, Ignore Case)

Return Value

Returns a Boolean. TRUE if the text is found, FALSE otherwise.

Example

Script( Search_Screen_Test )
String( strSearch )
Boolean( bFound )
Boolean( bIgnoreCase )
Number( nStartRow )
Number( nEndRow )
Activate( From_Menu )
    strSearch = Ask_String( "Enter string to search for", "Search_String_Test", 0, 999, "" )
    nStartRow = Ask_Number( "Enter starting row (1 is top of screen)
                        ", "Search_String_Test", 0, 999, 0 )
    bIgnoreCase = Ask_Yes_No( "Ignore case?", "Search_String_Test", FALSE )
    bFound = Search_Screen( strSearch, nStartRow, nEndRow, bIgnoreCase )
    If( bFound )
        Message( "Search string found.", 5 )
    Else
Message( "Search string not found.", 5 )
End_If
Return

See Also

Get_Screen_Rows, Get_Position_Row, Get_Field_row, Get_Field_Index, Set_Field_Data_ID,
String_Set, Ask_String
**Set_Cursor_Position**

Moves the cursor to the specified row and column. The top-most row is 1, and the left-most column is 1.

**Parameters**

- **Row** The row where the cursor will go, starting at row 1 for the top row.
- **Column** The column where the cursor will go, starting at column 1 for the left-most column.

**Format**

```
Set_Cursor_Position (Row, Column)
```

**Example**

```
Script(Set_Cursor_Position_Test)
    Activate(From_Menu)
    Set_Cursor_Position(15, 6)
    Return
```

**See Also**

`Keypress_String, Keypress_Key, Scan_String, Get_Screen_Columns, Get_Screen_Rows, Get_Position_Column, Get_Position_Row, Get_Field_Index, Get_Field_Row, Get_Field_Column`
**Set_Field_Append_Scan_Data**

Controls whether to append scan data in the field. This action is only valid when using IBM 5250 or 5555 emulation.

**Parameters**

- **Field Index**: The numeric index of the 5250 data field, index 0 is the first field.
- **Append Scan Data**: Use TRUE to cause the scan data to be appended to the scanned value when scanning a field.

**Format**

```
Set_Field_Append_Scan_Data (Field Index, Append Scan Data)
```

**Return Value**

Returns a Boolean. TRUE if successful, returns FALSE if the field index is not valid.

**Example**

```
Script(Set_Field_Append_Scan_Data_Test)
Boolean(bSetOK)
Activate(From_Menu)
    bSetOK = Set_Field_Append_Scan_Data(0, FALSE)
    If(bSetOK)
        Message("Set_Field_Append_Scan_Data worked", 5)
    Else
        Message("Set_Field_Append_Scan_Data failed", 5)
    End_If
    Return
```

**See Also**

- `Set_Field_Data_ID`, `Set_Field_Symbology_ID`, `Get_Field_Symbology_Operator`, `Set_Field_Com_Data_Field`, `Set_Field_Prefix_Scan_Data`, `Get_Field_Append_Scan_Data`, `Get_Field_Data_ID`, `Get_Num_Field_Data_IDs`, `Get_Num_Field_Symbology_IDs`, `Get_Field_Com_Data_Field`, `Get_Field_Symbology_ID`, `Get_Num_Fields`, `Get_Field_Prefix_Scan_Data`
**Set_Field_Com_Data_Field**

Sets a field to be the Com Data Field for the screen. There can be only one Com Data Field per screen. Use FALSE to remove the Com Data Field setting.

**NOTE:** This action is only valid when using IBM 5250 or 5555 emulation.

### Parameters

- **Field Index**
  
The numeric index of the 5250 data field, index 0 is the first field.

- **Set Com Data Field**
  
  Use TRUE to make this field a Com Data Field.

### Format

```
Set_Field_Com_Data_Field (Field Index, Set Com Data Field)
```

### Return Value

Returns a Boolean. TRUE if successful, FALSE if the index is not valid.

### Example

```script
Script(Set_Field_Com_Data_Field_Test)
Boolean(bSetOK)
Activate(From_Menu)
    bSetOK = Set_Field_Com_Data_Field(2, FALSE)
    If(bSetOK)
        Message("Set_Field_Com_Data_Field worked", 5)
    Else
        Message("Set_Field_Com_Data_Field failed", 5)
    End_If
Return
```

### See Also

- Set_Field_Data_ID, Set_Field_Symbology_ID, Get_Field_Symbology_Operator, Set_Field_Append_Scan_Data, Set_Field_Prefix_Scan_Data, Get_Field_Append_Scan_Data, Get_Field_Data_ID, Get_Num_Field_Data_IDs, Get_Num_Field_Symbology_IDs, Get_Field_Com_Data_Field, Get_Field_Symbology_ID, Get_Num_Fields, Get_Field_Prefix_Scan_Data
**Set_Field_Data_ID**

Sets the Data ID for a field. A field may have more than one Data ID. If the field already has a Data ID, this command will add another Data ID. Use a blank string to clear all Data IDs for the field.

Use actions like `Get_Field_Index()` to determine the index of a field.

**NOTE:** This action is only valid when using IBM 5250 or 5555 emulation.

### Parameters

- **Field Index**
  - The numeric index of the 5250 data field, index 0 is the first field.

- **Data ID String**
  - The data identifier for the 5250 data field, blank to clear all data identifiers for the field.

### Format

```plaintext
Set_Field_Data_ID (Field Index, "Data ID String")
```

### Return Value

Returns a Boolean. TRUE if successful, FALSE if the field index is not valid.

### Example

```plaintext
Script(Set_Field_Data_ID_Test)
Boolean(bSetOK)
Activate(From_Menu)
  bSetOK = Set_Field_Data_ID(0, "N")
  If(bSetOK)
    Message("Set_Field_Data_ID worked", 5)
  Else
    Message("Set_Field_Data_ID failed", 5)
  End_If
End_SCRIPT
```

### See Also

- `Set_Field_Symbology_ID`
- `Get_Field_Symbology_Operator`
- `Set_Field_Append_Scan_Data`
- `Set_Field_Com_Data_Field`
- `Set_Field_Prefix_Scan_Data`
- `Get_Field_Append_Scan_Data`
- `Get_Field_Com_Data_Field_ID`
- `Get_Num_Field_Data_IDs`
- `Get_Field_Symbology_ID`
- `Get_Field_Com_Data_Field`
- `Get_Field_Prefix_Scan_Data`
### Set_Field_Prefix_Scan_Data

Sets the data prefixed to a field when the field is scanned. Use a blank string to clear the prefix data.

**NOTE:** This action is only valid when using IBM 5250 or 5555 emulation.

#### Parameters

- **Field Index**: The numeric index of the 5250 data field, index 0 is the first field.
- **Data To Prefix**: Gets prefixed to the field data.

#### Format

```
Set_Field_Prefix_Scan_Data (Field Index, "Data To Prefix")
```

#### Return Value

Returns a Boolean. TRUE if successful, FALSE if the field index is not valid.

#### Example

```
Script(Set_Field_Prefix_Scan_Data_Test)
Boolean(bSetOK)
Activate(From_Menu)
  bSetOK = Set_Field_Prefix_Scan_Data(0, "99")
  If(bSetOK)
    Message("Set_Field_Prefix_Scan_Data worked", 5)
  Else
    Message("Set_Field_Prefix_Scan_Data failed", 5)
  End_If
Return
```

#### See Also

- `Set_Field_Data_ID`, `Set_Field_Symbology_ID`, `Get_Field_Symbology_Operator`, `Set_Field_Append_Scan_Data`, `Set_Field_Com_Data_Field`, `Get_Field_Append_Scan_Data`, `Get_Field_Data_ID`, `Get_Num_Field_Data_IDS`, `Get_Num_Field_Symbology_IDS`, `Get_Field_Com_Data_Field`, `Get_Field_Symbology_ID`, `Get_Num.Fields`, `Get_Field_Prefix_Scan_Data`
Set_Field_Symbology_ID

Sets the Symbology ID for a field. A field may have more than one Symbology ID. If the field already has a Symbology ID, this command will add another Symbology ID. Use Symbology ID ANY to clear the symbologies, which will then allow you to use All Symbologies. ANY causes Get_Num_Field_Symbology_IDS () to return zero, and Get_Field_Symbology_ID () to return an empty string.

NOTE: This action is only valid when using IBM 5250 or 5555 emulation.

Scanner symbology values can be found in Symbologies and Values on page 369.

Parameters

- **Field Index**: The numeric index of the 5250 data field, index 0 is the first field.
- **Symbology ID**: The name of the symbology.
- **And-Or with Data ID**: Indicates whether the field data must match the Data ID and/or Symbology ID.

Format

Set_Field_Symbology_ID (Field Index, "Symbology ID", And-Or with Data ID)

Return Value

If And-Or with Data ID is TRUE, then the field data must match both the Data ID and the Symbology ID. If And-Or is FALSE, then the field data must match either the Data ID or the Symbology ID.

Example

Script( Set_Field_Symbology_ID_Test )
Boolean( bSetOK )
Activate( From_Menu )
bSetOK = Set_Field_Symbology_ID( 2, "UPCA", FALSE )
If( bSetOK )
    Message( "Set_Field_Symbology_ID worked", 5 )
Else
    Message( "Set_Field_Symbology_ID failed", 5 )
End_If
Return
See Also

Set_Field_Data_ID, Get_Field_Symbology.Operator, Set_Field_Append.Scan_Data, Set_Field_Com_Data_Field, Set_Field_Prefix.Scan_Data, Get_Field_Append.Scan_Data, Get_Field_Data_ID, Get_Num_Field_Data_IDS, Get_Num_Field_Symbology_IDS, Get_Field_Com_Data_Field, Get_Field_Symbology_ID, Get_Num.Fields, Get_Field_Prefix.Scan_Data
Speech_Change_Setting

Changes the speech setting to the specified value.

If the return is FALSE and the setting starts with "stt", call Speech_To_Text_Error_Desc to get the reason for the error.

If the return is FALSE and the setting starts with "tts", call Speech_From_Text_Error_Desc to get the reason for the error.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setting</td>
<td>The name of the setting to change.</td>
</tr>
<tr>
<td>Value</td>
<td>The new value for the setting.</td>
</tr>
</tbody>
</table>

Format

Speech_Change_Setting ("Setting", Value)

Return Value

Returns a Boolean. TRUE if the setting is supported and the value is valid for that setting, returns FALSE otherwise.

Example

Script(Speech_Change_Setting_Test)
String( strDescription )
Boolean( bChanged )
Activate( From_Menu )
Comment: Increase the speech-to-text timeout to twenty seconds
  bChanged = Speech_Change_Setting("stt_timeout", 20000)
  If( bChanged )
    Message( "Changed OK", 3 )
  Else
    strDescription = Speech_To_Text_Error_Desc
    Message( String.Combine( "Change Failed: ", strDescription ), 5 )
  End_If
Return

See Also

Speech_From_Text_Available, Speech_From_Text, Speech_To_Text_Available, Speech_To_Text, Speech_To_Text_No_Wait, Speech_To_Text_Cancel, Speech_Setting_Available, Speech_
Get_Setting, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_Get_Setting_Value_Desc, Speech_To_Text_Get_User_Name, Speech_To_Text_Change_User_Name, Speech_From_Text_Error_Desc, Speech_To_Text_Error_Desc, Speech_From_Text_Cancel
Speech_Find_Setting_Value

Searches all possible value descriptions for the speech setting.

**Parameters**

- **Setting**: The name of the setting to search.
- **Value Description**: The value description to match.
- **Exact Only**: Indicates whether only an exact match is returned.

**Format**

```plaintext
Speech_Find_Setting_Value ("Setting", Value Description, Exactly Only)
```

**Return Value**

Returns the value of the setting that is the closest match. If "Exact Only" is TRUE, then only exact matches are returned. Returns -1 if no match is found.

**Example**

```plaintext
Script(Speech_Find_Setting_Value_Test)
String(strMessage)
Number(nLanguage)
Activate(From_Menu)
    nLanguage = Speech_Find_Setting_Value("stt_language", "enu", FALSE)
    strMessage = String_Combine("stt_language match for enu:", Number_To_String_Decimal(nLanguage))
Message(strMessage, 5)
Return
```

**See Also**

Speech_From_Text_Available, Speech_From_Text, Speech_To_Text_Available, Speech_To_Text, Speech_To_Text_No_Wait, Speech_To_Text_Cancel, Speech_Setting_Available, Speech_Change_Setting, Speech_Get_Setting, Speech_Get_Setting_Max, Speech_Get_Setting_Value_Desc, Speech_To_Text_Get_User_Name, Speech_To_Text_Change_User_NameSpeech_From_Text_Error_Desc, Speech_To_Text_Error_Desc, Speech_From_Text_Cancel
Speech_From_Text

Converts text into sound and plays the resulting sound on the computer. If the parameter Wait Until Done is FALSE, the script will continue to execute while the sound is being played.

For Temporary Settings, each setting must start with "tts_" and use the format setting=value. Multiple settings may be specified, and should be separated by comma (,) characters. Quotes around the value are optional. If a value is not a number, then the Text-to-Speech engine uses the value closest to the value text description provided. Temporary Settings are only in effect only while the speech is being played.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>The text that is converted into sound.</td>
</tr>
<tr>
<td>Wait Until Done</td>
<td>Indicates whether the script resumes execution before the speech has completely played.</td>
</tr>
<tr>
<td>Temporary Settings</td>
<td>An optional string to set one or more speech settings only for the duration of this Text-to-Speech action.</td>
</tr>
</tbody>
</table>

Format

Speech_From_Text ("Text", Wait Until Done)

or

Speech_From_Text ("Text", Wait Until Done, "Optional Temporary Settings")

Return Value

Returns a Boolean. TRUE if the sound was played successfully, FALSE if otherwise.

Example

```w
Script( Speech_From_Text_Test )
Activate( From_Menu )
    Speech_From_Text( "Hello again.", FALSE )
    Speech_From_Text( "A508421", FALSE, "tts_readmode=character, tts_rate=200" )
    Speech_From_Text( "Goodbye", FALSE )
Return
```
See Also

Speech_From_Text_Available, Speech_To_Text_Available, Speech_To_Text, Speech_To_Text_No_Wait, Speech_To_Text_Cancel, Speech_Setting_Available, Speech_Change_Setting, Speech_Get_Setting, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_Get_Setting_Value_Desc, Speech_To_Text_Get_User_Name, Speech_To_Text_Change_User_Name, Speech_From_Text_Error_Desc, Speech_To_Text_Error_Desc, Speech_From_Text_Cancel, Speech_Get_Confidence_Level
Speech_From_Text_Available

Determines whether text-to-speech is supported.

Return Value

Returns a Boolean. TRUE if text-to-speech is supported on the computer, returns FALSE otherwise.

Example

Script(Speech_From_Text_Available_Test)
Boolean(bAvailable)
Activate(From_Menu)
    bAvailable = Speech_From_Text_Available
    If(bAvailable)
        Message("Speech From Text is available", 5)
    Else
        Message("Speech From Text is not available", 5)
    End_If
Return

See Also

Speech_From_Text, Speech_To_Text_Available, Speech_To_Text, Speech_To_Text_No_Wait, Speech_To_Text_Cancel, Speech_Setting_Available, Speech_Change_Setting, Speech_Get_Setting, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_Get_Setting_Value_Desc, Speech_To_Text_Get_User_Name, Speech_To_Text_Change_User_Name, Speech_From_Text_Error_Desc, Speech_To_Text_Error_Desc, Speech_From_Text_Cancel
**Speech_From_Text_Cancel**

Provides a way for the script to perform other functions while the text-to-speech action occurs. Returns immediately if there is no action to cancel.

**Return Value**

Returns after canceling any unspoken speech-from-text actions.

**Example**

```plaintext
Script(Speech_From_Text_Cancel_Test)
Activate(From_Menu)
    Speech_From_Text("The quick brown fox jumped over the lazy dogs.", FALSE)
Delay(1000)
Speech_From_Text_Cancel
Return
```

**See Also**

Speech_From_Text_Available, Speech_From_Text, Speech_To_Text_Available, Speech_To_Text, Speech_To_Text_No_Wait, Speech_Setting_Available, Speech_Change_Setting, Speech_Get_Setting, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_Get_Setting_Value_DESC, Speech_To_Text_Get_User_Name, Speech_To_Text_Change_User_Name, Speech_To_Text_Cancel, Speech_Get_Confidence_Level
Speech_From_Text_Error_Desc

Gets an error description for the last speech-from-text action. An empty string is returned if no errors have occurred.

Return Value

Returns a string describing the last error from a speech-from-text action.

Example

```w"Script( Speech_From_Text_Error_Desc_Test )
String( strDescription )
Activate( From_Menu )
    strDescription = Speech_From_Text_Error_Desc
    Message( String.Combine( "Last speech error:", strDescription ), 5 )
Return```

See Also

Speech_From_Text_Available, Speech_From_Text, Speech_To_Text_Available, Speech_To_Text, Speech_To_Text_No_Wait, Speech_Setting_Available, Speech_Change_Setting, Speech_Get_Setting, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_Get_Setting_Value_Desc, Speech_To_Text_Get_User_Name, Speech_To_Text_Chang
**Speech_Get_Confidence_Level**

Gets the confidence value for the last Speech_To_Text action.

**Return Value**

Returns a number that is the confidence value for the last successful Speech_To_Text result. The number is zero if the confidence value is not available.

**Example**

```plaintext
Script( Speech_Get_Confidence_Level_Test )
String( szResult )
String( szMessage )
Number( nConfidence )
Activate( From_Menu )
    If_Not( Speech_To_Text_Available )
        Ask_OK( "Speech to text is not available.", "Error" )
        Return
    End_If
    Message( "Say one or more digits", 0 )
    If_Not( Speech_To_Text( szResult, "connected_digits" ) )
        Message_Clear
        Ask_OK( "No results returned from the speech.", "Error" )
        Return
    End_If
    Message_Clear
    nConfidence = Speech_Get_Confidence_Level
    szMessage = String_Combine( szResult, ", confidence: " )
    szMessage = String_Combine( szMessage, Number_To_String_Decimal( nConfidence ) )
    Ask_OK( szMessage, "Speech-to-text-confidence" )
Return
```

**See Also**

*Speech_From_Text, Speech_From_Text_Cancel*
Chapter 7: Overview of Actions

Speech_Get_Setting

Gets the value of the speech setting.

Parameters

Setting  Get the value for this setting name.

Format

Speech_Get_Setting ("Setting")

Return Value

Returns the current value of the speech setting. Returns -1 if the speech setting is not valid.

Example

Script(Speech_Get_Setting_Test)
String(strTimeout)
Number(nTimeout)
Activate(From_Menu)
  nTimeout = Speech_Get_Setting("stt_timeout")
  strTimeout = Number_To_String_Decimal(nTimeout)
  Message(strTimeout, 3)
Return

See Also

Speech_From_Text_Available, Speech_From_Text, Speech_To_Text_Available, Speech_To_Text, Speech_To_Text_No_Wait, Speech_To_Text_Cancel, Speech_Setting_Available, Speech_Change_Setting, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_Get_Setting_Value_Desc, Speech_To_Text_Get_User_Name, Speech_To_Text_Change_User_Name, Speech_From_Text_Error_Desc, Speech_To_Text_Error_Desc, Speech_From_Text_Cancel
## Speech_Get_Setting_Max

Gets the largest value for a speech setting.

### Parameters

- **Setting** Get the maximum value for this setting name.

### Format

```
Speech_Get_Setting_Max ("Setting")
```

### Return Value

Returns the largest possible value for a speech setting. Returns 0 if only one setting value is supported, returns -1 if the speech setting is not valid.

### Example

```
Script(Speech_Get_Setting_Max_Test)
String(strMessage)
Number(nTimeout)
Activate(From_Menu)
    nTimeout = Speech_Get_Setting_Max("stt_timeout")
    strMessage = String_Combine("stt_timeout maximum:", Number_To_String_Decimal(nTimeout))
Message(strMessage, 5)
Return
```

### See Also

- [Speech_From_Text_Available](#), [Speech_From_Text](#), [Speech_To_Text_Available](#), [Speech_To_Text](#), [Speech_To_Text_No_Wait](#), [Speech_To_Text_Cancel](#), [Speech_Setting_Available](#), [Speech_Change_Setting](#), [Speech_Get_Setting](#), [Speech_Find_Setting_Value](#), [Speech_Get_Setting_Value_Desc](#), [Speech_To_Text_Get_User_Name](#), [Speech_To_Text_Change_User_Name](#), [Speech_From_Text_Error_Desc](#), [Speech_To_Text_Error_Desc](#), [Speech_From_Text_Cancel](#)
**Speech_Get_Setting_Value_Desc**

Get a description of the speech setting value.

**Parameters**

- **Setting**  The name of the setting.
- **Value**    The description for this value.

**Format**

```
Speech_Get_Setting_Value_Desc ("Setting", Value)
```

**Return Value**

Returns a string that describes the value for the speech setting (this does not need to be the setting’s current value). Returns an empty string if the setting or value is not valid.

**Example**

```wisp
Script( Speech_Get_Setting_Value_Desc_Test )
String( strDescription )
Activate( From Menu )
    strDescription = Speech_Get_Setting_Value_Desc( "stt_language", 1 )
Message( strDescription, 7 )
Return
```

**See Also**

Speech_From_Text_Available, Speech_From_Text, Speech_To_Text_Available, Speech_To_Text, Speech_To_Text_No_Wait, Speech_To_Text_Cancel, Speech_Setting_Available, Speech_Change_Setting, Speech_Get_Setting, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_To_Text_Get_User_Name, Speech_To_Text_Change_User_Name, Speech_From_Text_Error_Desc, Speech_To_Text_Error_Desc, Speech_From_Text_Cancel
Speech_Setting_Available

Identifies speech settings by case-insensitive name strings. For a list of available setting names, see Speakeasy Settings on page 309.

Parameters

Setting   Check if this setting name is supported.

Format

Speech_Setting_Available (Setting)

Return Value

Returns a Boolean. TRUE if the speech setting name is supported, FALSE otherwise.

Example

Script(Speech_Setting_Available_Test)
Boolean(bAvailable)
   bAvailable = Speech_Setting_Available("stt_language")
   If(bAvailable)
       Message("Speech setting stt_language is available", 5)
   Else
       Message("Speech setting stt_language is not available", 5)
   End_If
Return

See Also

Speech_From_Text_Available, Speech_From_Text, Speech_To_Text_Available, Speech_To_Text, Speech_To_Text_No_Wait, Speech_To_Text_Cancel, Speech_Change_Setting, Speech_Get_Setting, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_Get_Setting_Value_Desc, Speech_To_Text_Get_User_Name, Speech_To_Text_Change_User_Name, Speech_From_Text_Error_Desc, Speech_To_Text_Error_Desc, Speech_From_Text_Cancel
Speech_To_Text

Listens to the user speak and returns the text equivalent of what he/she said in the string variable. If a grammar is specified, the grammar file with that name is used for speech recognition; otherwise, the previous grammar file is reused.

**Parameters**

- **Text**  
The string variable into which the converted speech is placed.
- **Grammar**  
The grammar file to use.

**Return Value**

Returns a string, the text equivalent of a user’s speech. Returns an empty string if no acceptable speech was detected. An empty string result may indicate a timeout condition or an error condition.

**Example**

```plaintext
Script( Speech_To_Text_Test )
String( szResult )
Activate( From_Menu )
  If_Not( Speech_To_Text_Available )
    Ask_OK( "Speech to text is not available.", "Error" )
    Return
  End_If
Message( "Say one or more digits", 0 )
If_Not( Speech_To_Text( szResult, "connected_digits" ) )
  Message_Clear
  Ask_OK( "No results returned from the speech.", "Error" )
  Return
End_If
Message_Clear
Ask_OK( szResult, "Speech-to-text-results" )
Return
```

**See Also**

Speech_From_Text_Available, Speech_From_Text, Speech_To_Text_Available, Speech_To_Text_No_Wait, Speech_To_Text_Cancel, Speech_Setting_Available, Speech_Change_Setting, Speech_Get_Setting, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_Get_Setting_Value_Desc, Speech_To_Text_Get_User_Name, Speech_To_Text_Change_User_Name, Speech_From_Text_Error_Desc, Speech_To_Text_Error_Desc, Speech_From_Text_Cancel
Chapter 7: Overview of Actions

Speech_To_Text_Available

Determines whether speech-to-text is supported.

Return Value

Returns a Boolean. TRUE if speech-to-text is supported on the computer, returns FALSE otherwise.

Example

Script(Speech_To_Text_Available_Test)
Boolean(bAvailable)
Activate(From_Menu)
    bAvailable = Speech_To_Text_Available
    If(bAvailable)
        Message("Speech To Text is available", 5)
    Else
        Message("Speech To Text is not available", 5)
    End_If
Return

See Also

Speech_From_Text_Available, Speech_From_Text, Speech_To_Text, Speech_To_Text_No_Wait, Speech_To_Text_Cancel, Speech_Setting_Available, Speech_Change_Setting, Speech_Get_Setting, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_Get_Setting_Value_Desc, Speech_To_Text_Get_User_Name, Speech_To_Text_Change_User_Name, Speech_From_Text_Error_Desc, Speech_To_Text_Error_Desc, Speech_From_Text_Cancel
Chapter 7: Overview of Actions

Speech_To_Text_Cancel

Provides a way for the script to perform other functions while the speech-to-text action occurs.

Return Value

Returns after cancelling the last Speech_To_Text_No_Wait action. Returns immediately if there is no action to cancel.

Example

See the example for Speech_To_Text_No_Wait.

See Also

Speech_From_Text_Available, Speech_From_Text, Speech_To_Text_Available, Speech_To_Text, Speech_To_Text_No_Wait, Speech_Setting_Available, Speech Change Setting, Speech_Get_Setting, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_Get_Setting_ Value_Desc, Speech_To_Text_Get_User_Name, Speech_To_Text_Change_User_Name, Speech_From_Text_Error_Desc, Speech_To_Text_Error_Desc, Speech_From_Text_Cancel
**Speech_To_Text_Change_User_Name**

Changes the user name being used by the speech-to-text engine.

### Return Value

Returns a Boolean. **TRUE** if the user name was changed, **FALSE** if this feature is not supported.

### Example

```plaintext
Script( Speech_To_Text_Change_User_Name_Test )
String( strUsername )
Activate( From_Menu )
  strUsername = Ask_String( "Enter the new user name", "New User Name", 1,25, "" )
  Speech_To_Text_Change_User_Name( strUsername )
Return
```

### See Also

- `Speech_From_Text_Available`, `Speech_From_Text`, `Speech_To_Text_Available`, `Speech_To_Text`, `Speech_To_Text_No_Wait`, `Speech_To_Text_Cancel`, `Speech_Setting_Available`, `Speech_Change_Setting`, `Speech_Get_Setting`, `Speech_Get_Setting_Max`, `Speech_Find_Setting_Value`, `Speech_Get_Setting_Value_Desc`, `Speech_To_Text_Get_User_Name`, `Speech_From_Text_Error_Desc`, `Speech_To_Text_Error_Desc`, `Speech_From_Text_Cancel`
**Speech_To_Text_Error_Desc**

Gets an error description for the last speech-to-text action. An empty string is returned if no errors have occurred.

**Return Value**

Returns a string describing the last error from a speech-to-text action.

**Example**

```plaintext
Script( Speech_To_Text_Error_Desc_Test )
String( strDescription )
Activate( From_Menu )
Speech_Change_Setting( "stt_unknown", 2 )
strDescription = Speech_To_Text_Error_Desc
Message( String_Combine( "Last speech-to-text error:\n", strDescription),5 )
Return
```

**See Also**

Speech_From_Text_Available, Speech_From_Text, Speech_To_Text_Available, Speech_To_Text, Speech_To_Text_No_Wait, Speech_Setting_Available, Speech_Change_Setting, Speech_Get_Setting, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_Get_Setting_Value_Desc, Speech_To_Text_Get_User_Name, Speech_To_Text_Change_User_Name, Speech_From_Text_Error_Desc, Speech_From_Text_Cancel
Speech_To_Text_Get_User_Name

Gets the user name.

Return Value

Returns a string with the user name being used by the speech-to-text engine. An empty string is returned if no user name has been assigned.

Example

```wscript
Script( Speech_To_Text_Get_User_Name_Test )
String( strUsername )
Activate( From_Menu )
    strUsername = Speech_To_Text_Get_User_Name
Message( strUsername, 4 )
Return
```

See Also

Speech_From_Text_Available, Speech_From_Text, Speech_To_Text_Available, Speech_To_Text, Speech_To_Text_No_Wait, Speech_To_Text_Cancel, Speech_Setting_Available, Speech_Change_Setting, Speech_Get_Setting, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_Get_Setting_Value_Desc, Speech_To_Text_Change_User_Name, Speech_From_Text_Error_Desc, Speech_To_Text_Error_Desc, Speech_From_Text_Cancel
Speech_To_Text_No_Wait

Listens to the user speak and returns the text equivalent of what the user said. The Boolean variable is set to TRUE when the speech is recognized or times out. If a grammar is specified, the grammar file with that name is used for the speech recognition. Otherwise, the previous grammar file is reused.

**Parameters**

- **Done** Indicates when the speech is recognized or times out.
- **Text** The string variable into which the converted speech is placed.
- **Grammar** The grammar file to use.

**Format**

```plaintext
Speech_To_Text_No_Wait (Done, Text, Grammar)
```

**Return Value**

Returns the text equivalent of a user’s speech in the string variable. Returns a Boolean value of either TRUE or FALSE. It returns TRUE if the speech-to-text started successfully, or FALSE otherwise.

**Example**

```plaintext
Script( Speech_Demo )
Profile( Speech_Demo_Profile )
String( sSpeechResult )
Boolean( bSpeechStarted )
Boolean( bSpeechDone )
Number( nCurrentGrammar )
Number( nDesiredGrammar )
Number( nCurrentScreen )
Activate( Connection )

Comment: Is Speech available?
If_Not( Speech_To_Text_Available )
    Message( "Speech To Text Not available", 3 )
    Return
End_If
If_Not( Speech_From_Text_Available )
    Message( "Speech From Text Not Available", 3 )
    Return
End_If
```
Comment: We are using English.

Speech_Change_Setting( "stt_language_long", Speech_Find_Setting_Value( "stt_language_long", "English", FALSE ))
Speech_Change_Setting( "tts_language_long", Speech_Find_Setting_Value("tts_language_long", "English", FALSE ))

While( TRUE )

Comment: Perform Speech-to-Text with the desired grammar.
If( Number_Not_Equal( nCurrentGrammar, nDesiredGrammar ) )
   Speech_To_Text_Cancel
   bSpeechStarted = FALSE
   bSpeechDone = FALSE
   sSpeechResult = ""
End_If
If( Boolean_And( bSpeechDone, String.Empty( sSpeechResult ) ) )
   Comment: The string was cleared because the result was used. Reset for the next use.
   bSpeechStarted = FALSE
   bSpeechDone = FALSE
End_If
If_Not( bSpeechStarted )
   If( Number_Equal( nDesiredGrammar, 1 ) )
      Comment: Set the threshold for this grammar.
      Comment: Lower values are more likely to get results, but they are more likely to be wrong.
      Speech_Change_Setting( "stt_threshold", 4500 )

      Comment: Use the digit.bnf grammar file.
      bSpeechStarted = Speech_To_Text_No_Wait( bSpeechDone,
      sSpeechResult, "digit" )
   End_If

   Comment: Can add support for other grammars here.

   nCurrentGrammar = nDesiredGrammar
End_If

Comment: Look for screens where we include speech support.
   If( Boolean_And( String_Equal( Get_Screen_Text_Length( 1, 1, 6 ),
      "Pick ", 0, FALSE ), String_Equal( Get_Screen_Text_Length( 7, 1, 4 ),
      "Pick", 0, FALSE ) ) )
   Comment: The first time we see this screen, tell the user what we want.
   If( Number_Not_Equal( nCurrentScreen, 101 ) )
nCurrentScreen = 101
Speech_From_Text( "Pick a menu item from 1 to 5.", FALSE )
End_If

Comment: Prepare to get the user's response.

nDesiredGrammar = 1
If( Number_Not_Equal( nDesiredGrammar, nCurrentGrammar ) )
Continue
End_If

Comment: Handle any user responses.

If_Not( String.Empty( sSpeechResult ) )
If( Boolean_And( Number_Greater_Than_Or_Equal( String_To_Number_Decimal( sSpeechResult ), 1 ), Number_Less_Than_Or_Equal( String_To_Number_Decimal( sSpeechResult ), 5 ) ) )
Comment: Type the response the user supplied.
Keypress_String( sSpeechResult )
Keypress_Key( "VT220", "Enter" )
Wait_For_Screen_Update
nDesiredGrammar = 0
sSpeechResult = ""
Continue
End_If

Speech_From_Text( "Unexpected result. Please try again.", FALSE )
sSpeechResult = ""
Continue
End_If

Comment: Wait for the user to respond.
Wait_For_Screen_Update
Continue
End_If

Comment: Can add support for other screens here.

Comment: If we reach this point, we don't recognize the current screen.
Comment: Wait for a screen we recognize.
nCurrentScreen = 0
nDesiredGrammar = 0
Wait_For_Screen_Update
End_While
Return

See Also

String.Combine

Returns the concatenated value of two to five strings.

**Parameters**

- String1: The first part of the returned string.
- String2: The second part of the returned string.
- String3: The optional third part of the returned string.
- String4: The optional fourth part of the returned string.
- String5: The optional fifth part of the returned string.

**Format**

```
String.Combine (String1, String2)
```

**Return Value**

Returns the value of String1 concatenated with String2. The optional strings are concatenated if they are present.

**Example**

```wolfram
Script[String_Set_Test]
String[strResult]
String[strTitle]
Activate[From_Menu]
    strTitle = "Scripting String"
    strResult = Ask_String("Enter some text", "String_Set_Test", 1, 99, "")
    strResult = String.Combine("Text Entered:", strResult)
    Ask_OK[strResult, strTitle]
Return
```

**See Also**

String_Empty

Check the length of the string to determine if it is an empty string.

Parameters

String  Check if this string is empty.

Format

String_Empty (String)

Return Value

Returns Boolean. TRUE if the string is 0 characters in length, FALSE otherwise.

Example

Script(String_Empty_Test)
String(strEntered)
Boolean(bEmpty)
Activate(From_Menu)
    strEntered = Ask_String("Type a string", "String_Empty_Test", 0, 0, "")
    bEmpty = String_Empty(strEntered)
    If(bEmpty)
        Message("String is empty.", 5)
    Else
        Message("String is not empty.", 5)
    End_If
Return

See Also

String_Less_Than, String_Less_Than_Or_Equal, String_Equal, String_Greater_Than_Or_Equal, String_Greater_Than, String_Not_Equal, String_Set, Number_To_String.Decimal, Ask_String, Get_Screen_Text, Speech_To_Text, Get_Scan_Type_Name
String_Equal

Compare the two strings and determine if they are both TRUE or are both FALSE. If the Maximum Length value is greater than 0, any characters after the specified number of characters are ignored. If Ignore Case is TRUE, then upper-case and lower-case letters are considered to be equal.

Parameters

- **Test1** Gets compared with Test2.
- **Test2** Gets compared with Test1.
- **Maximum Length** Indicates whether the number of characters is limited.
- **Ignore Case** Indicates whether the case of the letters is taken into consideration.

Format

String_Equal ("Test1", "Test2", Maximum Length, Ignore Case)

Return Value

Returns a Boolean. TRUE if Test1 precedes Test2 are the same string, FALSE otherwise.

Example

Script(String_Equal_Test)
String(strEntered1)
String(strEntered2)
Boolean(bEqual)
Activate(From_Menu)
    strEntered1 = Ask_String("Type the first string", "String_Equal_Test", 0, 0, "")
    strEntered2 = Ask_String("Type the second string", "String_Equal_Test", 0, 0, ")
    bEqual = String_Equal(strEntered1, strEntered2, 0, TRUE)
    If(bEqual)
        Message("First string is equal to second string", 5)
    Else
        Message("First string is not equal to second string", 5)
End_If
Return
See Also

String_Empty, String_Less_Than, String_Less_Than_Or_Equal, String_Greater_Than_Or_Equal, String_Greater_Than, String_Not_Equal, String_Set, Number_To_String_Decimal, Ask_String, Get_Screen_Text, Speech_To_Text, Get_Scan_Type_Name
Chapter 7: Overview of Actions

**String_Find_First**

Finds the first instance of the substring inside the string, and returns the position where that substring starts. The left-most position is 0, so a value of 0 would be returned if the string started with the substring. A value of -1 is returned if no instances of the substring are in the string.

**Parameters**

- **String to Parse**
  - The string that gets searched.
- **Substring to Find**
  - The instance of the substring to find in the parsed string.
- **Ignore Case**
  - Indicates whether the case of the letters is taken into consideration.

**Format**

`String_Find_First (String to Parse, Substring to Find, Ignore Case)`

**Example**

```wolfram
Script( String_Find )
String( strOriginal )
String( strMessage )
String( strTitle )
String( strSearchFor )
Number( nFirstIndex )
Number( nLastIndex )
Activate( From_Menu )
strOriginal = Ask_String( "Enter a string to search", "String_Find", 1, 99, "abcdef ABCDE" )
strSearchFor = String_Find_First( strOriginal, strSearchFor, TRUE )
nFirstIndex = String_Find_Last( strOriginal, strSearchFor, TRUE )
strTitle = String_Combine( "Search "", strOriginal )
strTitle = String_Combine( strTitle, "" for "" )
strTitle = String_Combine( strTitle, """" )
strMessage = String_Combine( "String_Find_First: ", Number_To_String_DECIMAL( nFirstIndex ) )
Ask_OK( strMessage, strTitle )
strMessage = String_Combine( "String_Find_Last: ", Number_To_String_DECIMAL( nLastIndex ) )
Ask_OK( strMessage, strTitle )
```
Return

See Also

String_Length, String_Find_Last, String_Equal, String_Set
**String_Find_Last**

Finds the last instance of the substring inside the string, and returns the position where that substring starts. The left-most position is 0, so a value of 0 would be returned if the string started with the substring. A value of -1 is returned if no instances of the substring are in the string.

**Parameters**

- **String to Parse** The string that gets searched.
- **Substring to Find** The instance of the substring to find in the parsed string.
- **Ignore Case** Indicates whether the case of the letters is taken into consideration.

**Format**

`String_Find_Last (String to Parse, Substring to Find, Ignore Case)`

**Example**

```plaintext
Script( String_Find )
String( strOriginal )
String( strMessage )
String( strTitle )
String( strSearchFor )
Number( nFirstIndex )
Number( nLastIndex )
Activate( From_Menu )
  strOriginal = Ask_String( "Enter a string to search", "String_Find", 1, 99, "abcdef ABCDE" )
  strSearchFor = Ask_String("Search for:", "String_Find", 1, 99, "bc")
  nFirstIndex = String_Find_First( strOriginal, strSearchFor, TRUE )
  nLastIndex = String_Find_Last( strOriginal, strSearchFor, TRUE )
  strTitle = String_Combine( "Search """, strOriginal )
  strTitle = String_Combine( strTitle, "" for """" )
  strTitle = String_Combine( strTitle, strSearchFor )
  strTitle = String_Combine( strTitle, """" )
  strMessage = String_Combine( "String_Find_First: ", Number_To_String_Decimal( nFirstIndex ) )
  Ask_OK( strMessage, strTitle )
  strMessage = String_Combine( "String_Find_Last: ", Number_To_String_Decimal( nLastIndex ) )
  Ask_OK( strMessage, strTitle )
Return
```
See Also

String_Length, String_Find_First, String_EQUAL, String_Set
String_Greater_Than

Compare the two strings and determine whether one follows the other in alphabetical order. If the **Maximum Length** value is greater than 0, any characters after the specified number of characters are ignored. If **Ignore Case** is TRUE, then upper-case and lower-case letters are considered to be equal.

**Parameters**

- *Test1* Gets compared with *Test2*.
- *Test2* Gets compared with *Test1*.
- *Maximum Length* Indicates whether the number of characters is limited.
- *Ignore Case* Indicates whether the case of the letters is taken into consideration.

**Format**

```
String_Greater_Than ("Test1", "Test2", Maximum Length, Ignore Case)
```

**Return Value**

Returns a Boolean. TRUE if *Test1* follows *Test2* in alphabetical ordering, FALSE otherwise.

**Example**

```wolfram
Script( String_Greater_Than_Test )
String( strEntered1 )
String( strEntered2 )
Boolean( bGreaterThan )
Activate( From_Menu )
    strEntered1 = Ask_String( "Type the first string", "String_Greater_Than_Test", 0, 0, "" )
    strEntered2 = Ask_String( "Type the second string", "String_Greater_Than_Test", 0, 0, "" )
    bGreaterThan = String_Greater_Than( strEntered1, strEntered2, 0, TRUE )
    If( bGreaterThan )
        Message( "First string is greater than second string", 5 )
    Else
        Message( "First string is not greater than second string", 5 )
End_If
Return
```
See Also

String_Empty, String_Less_Than, String_Less_Than_Or_Equal, String_Equal, String_Greater_Than_Or_Equal, String_Not_Equal, String_Set, Number_To_String.Decimal, Ask_String, Get_Screen_Text, Speech_To_Text, Get_Scan_Type_Name
String_Greater_Than_Or_Equal

Compare the two strings and determine whether one follows the other in alphabetical order, or they are the same string. If the Maximum Length value is greater than 0, any characters after the specified number of characters are ignored. If Ignore Case is TRUE, then upper-case and lower-case letters are considered to be equal.

Parameters

- **Test1**: Gets compared with Test2.
- **Test2**: Gets compared with Test1.
- **Maximum Length**: Indicates whether the number of characters is limited.
- **Ignore Case**: Indicates whether the case of the letters is taken into consideration.

Format

String_Greater_Than_Or_Equal ("Test1", "Test2", Maximum Length, Ignore Case)

Return Value

Returns a Boolean. TRUE if Test1 precedes Test2 in alphabetical ordering or they are the same string, FALSE otherwise.

Example

Script(String_Greater_Than_Or_Equal_Test)
String(strEntered1)
String(strEntered2)
Boolean(bGreaterThan)
Activate(From_Menu)
    strEntered1 = Ask_String("Type the first string", "String_Greater_Than_Or_Equal_Test", 0, 0, ")
    strEntered2 = Ask_String("Type the second string", "String"Greater_Than_Or_Equal_Test", 0, 0, ")
    bGreaterThan = String_Greater_Than_Or_Equal(strEntered1, strEntered2, 0, TRUE)
    If(bGreaterThan)
        Message("First string is greater than second string", 5)
    Else
        Message("First string is not greater than second string", 5)
End_If
Return
See Also

String_Empty, String_Less_Than, String_Less_Than_Or_Equal, String_Equal, String_Greater_Than, String_Not_Equal, String_Set, Number_To_String_Decimal, Ask_String, Get_Screen_Text, Speech_To_Text, Get_Scan_Type_Name
String_Left

Returns the specified characters of the input string.

Parameters

- **String**: The string from which to get the characters.
- **Number of Characters**: The number of characters to get, starting at the beginning of the string.

Format

```
String_Left (String, Number of Characters)
```

Return Value

Returns a string with just the first n characters of the input string. If the input string is less than n characters, the entire string is returned.

Example

```script
Script(String_Left_Test)
String(strEntered)
String(strLeftPart)
Activate(From_Menu)
    strEntered = Ask_String("Enter some text", "String_Left_Test", 1, 99, 
"
)
    strLeftPart = String_Left(strEntered, 2)
    Ask_OK(strLeftPart, "String_Left 2 characters")
Return
```

See Also

- String_Set, String_Combine, String_Right, String_Middle, String_Upper, String_Lower, String_Replace, String_Only_Characters, String_Strip_Characters, String_Trim_Spaces_Start, String_Trim_Spaces_End, Number_To_String_Binary, Number_To_String_Octal, Number_To_String_Decimal, Number_To_String_Hexadecimal_Lowercase, Number_To_String_Hexadecimal_Uppercase, Ask_String, Ask_String_Password, Ask_String_Uppercase, Ask_String_Lowercase, String_Equal
**String_Length**

Get the number of characters in a string.

**Parameters**

*String*  The string whose length is to be determined.

**Format**

```
String_Length (String)
```

**Return Value**

Returns the number of characters in the string. Returns 0 if the string is empty (has no characters).

**Example**

```
Script( String_Length_Test )
String( strForLength )
String( strTitle )
Number( nStringLength )
Activate( From_Menu )
    strForLength = Ask_String( "Enter a string", "String_Length", 1, 99, "abcde" )
    nStringLength = String_Length( strForLength )
    strTitle = String_Combine( "Length of ", strForLength )
    strTitle = String_Combine( strTitle, "" )
    Ask_OK( Number_To_String_Decimal( nStringLength ), strTitle )
Return
```

**See Also**

String_Find_First, String_Find_Last, String_Equal, String_Set
**String_Less_Than**

Compare the two strings and determine their alphabetical order. If the Maximum Length value is greater than 0, any characters after the specified number of characters are ignored. If Ignore Case is TRUE, then upper-case and lower-case letters are considered to be equal.

### Parameters

- **Test1** Gets compared with Test2.
- **Test2** Gets compared with Test1.
- **Maximum Length** Indicates whether the number of characters is limited.
- **Ignore Case** Indicates whether the case of the letters is taken into consideration.

### Format

```
String_Less_Than ("Test1", "Test2", Maximum Length, Ignore Case)
```

### Return Value

Returns a Boolean. TRUE if Test1 precedes Test2 in alphabetical ordering, FALSE otherwise.

### Example

```plaintext
Script(String_Less_Than_Test)
String(strEntered1)
String(strEntered2)
Boolean(bLessThan)
Activate(From_Menu)
    strEntered1 = Ask_String("Type the first string", "String_Less_Than_Test", 0, 0, ")
    strEntered2 = Ask_String("Type the second string", "String_Less_Than_Test", 0, 0, ")
    bLessThan = String_Less_Than(strEntered1, strEntered2, 0, TRUE)
    If(bLessThan)
        Message("First string is less than second string", 5)
    Else
        Message("First string is not less than second string", 5)
    End_If
Return
```
See Also

String_Empty, String_Less_Than_Or_Equal, String_Equal, String_Greater_Than_Or_Equal,
String_Greater_Than, String_Not_Equal, String_Set, Number_To_String_Decimal, Ask_String,
Get_Screen_Text, Speech_To_Text, Get_Scan_Type_Name
**String_Less_Than_Or_Equals**

Compare the two strings and determine whether one precedes the other in alphabetical order, or they are the same string. If the Maximum Length value is greater than 0, any characters after the specified number of characters are ignored. If Ignore Case is TRUE, then upper-case and lower-case letters are considered to be equal.

### Parameters

- **Test1**: Gets compared with Test2.
- **Test2**: Gets compared with Test1.
- **Maximum Length**: Indicates whether the number of characters is limited.
- **Ignore Case**: Indicates whether the case of the letters is taken into consideration.

### Format

```plaintext
String_Less_Than_Or_Equals ("Test1", "Test2", Maximum Length, Ignore Case)
```

### Return Value

Returns a Boolean. TRUE if Test1 precedes Test2 in alphabetical ordering or they are the same string, FALSE otherwise.

### Example

```plaintext
Script(String_Less_Than_Or_Equals_Test)
String(strEntered1)
String(strEntered2)
Boolean(bLessThan)
Activate(From_Menu)
    strEntered1 = Ask_String("Type the first string", 
                        "String_Less_Than_Or_Equals_Test", 0, 0, "")
    strEntered2 = Ask_String("Type the second string", 
                        "String_Less_Than_Or_Equals_Test", 0, 0, "")
    bLessThan = String_Less_Than_Or_Equals(strEntered1, strEntered2, 0, TRUE)
    If(bLessThan)
        Message("First string is less than or equal to second string", 5)
    Else
        Message("First string is not less than or equal to second string", 5)
```
End If
Return

See Also
String_Empty, String_Less_Than, String_Equal, String_Greater_Than_Or_Equal, String_Greater_Than, String_Not_Equals, String_Set, Number_To_String_Decimal, Ask_String, Get_Screen_Text, Speech_To_Text, Get_Scan_Type_Name
String_Lower

Converts the specified text to lowercase letters.

**Parameters**

String  The string to convert to lowercase.

**Format**

String_Lower (String)

**Return Value**

Returns a string with all characters converted to lowercase.

**Example**

```script
Script(String_Upper_Lower_Test)
String(strEntered)
String(strUpper)
String(strLower)
Activate(From_Menu)
    strEntered = Ask_String("Enter some text", "String_Upper_Lower_Test", 1, 99, "")
    strUpper = String_Upper(strEntered)
    Ask_OK(strUpper, "String_Upper_Test")
    strLower = String_Lower(strEntered)
    Ask_OK(strLower, "String_Lower_Test")
Return
```

**See Also**

String_Set, String_Combine, String_Left, String_Right, String_Middle, String_Upper, String_Replace, String_Only_Characters, String_Strip_Characters, String_Trim_Spaces_Start, String_Trim_Spaces_End, Number_To_String_Binary, Number_To_String_Octal, Number_To_String_Decimal, Number_To_String_Hexadecimal_Lowercase, Number_To_String_Hexadecimal_Uppercase, Ask_String, Ask_String_Password, Ask_String_Uppercase, Ask_String_Lowercase, String(Equal)
## String_Middle

Returns the specified characters of the input string. The string parsing starts at the position specified, with 0 being the left-most character, so a position value of 0 is the same as \( \text{String}_\text{Left} \).

### Parameters

- **String**: The string from which to get the characters.
- **Starting Position**: The character with which to start.
- **Number of Characters**: The number of characters to get from the end of the string.

### Format

\[
\text{String}_\text{Middle} \ (\text{String}, \ \text{Starting Position}, \ \text{Number of Characters})
\]

### Return Value

Returns a string with just the middle \( n \) characters of the input string. If the input string is less than \( n \) characters, the entire string is returned.

### Example

```plaintext
Script(String_Middle_Test)
String(strEntered)
String(strMiddlePart)
Number(nStart)
Number(nCharacters)
Activate(From_Menu)

    strEntered = Ask_String("Enter some text", "String_Middle_Test", 1, 99, "")
    nStart = Ask_Number("Enter start characters, zero is the first character", "String_Middle_Test", 0, 99, 2)
    nCharacters = Ask_Number("Enter number of characters", "String_Middle_Test", 0, 99, 3)
    strMiddlePart = String_Middle(strEntered, nStart, nCharacters)
Ask_OK(strMiddlePart, "String_Middle")
Return
```

### See Also

- \( \text{String}_\text{Set} \)
- \( \text{String}_\text{Combine} \)
- \( \text{String}_\text{Left} \)
- \( \text{String}_\text{Right} \)
- \( \text{String}_\text{Upper} \)
- \( \text{String}_\text{Lower} \)
- \( \text{String}_\text{Replace} \)
- \( \text{String}_\text{Only}\_\text{Characters} \)
- \( \text{String}_\text{Strip}\_\text{Characters} \)
- \( \text{String}_\text{Trim}\_\text{Spaces}\_\text{Start} \)
- \( \text{String}_\text{Trim}\_\text{Spaces}\_\text{End} \)
Trim_Spaces_End, Number_To_String_Binary, Number_To_String_Octal, Number_To_String_Decimal, Number_To_String_Hexadecimal_Lowercase, Number_To_String_Hexadecimal_Uppercase, Ask_String, Ask_String_Password, Ask_String_Uppercase, Ask_String_Lowercase, String_Equal
String_Not_Equal

Compare the two strings and return TRUE if they do not have the same value. If the Maximum Length value is greater than 0, any characters after the specified number of characters are ignored. If Ignore Case is TRUE, then upper-case and lower-case letters are considered to be equal.

Parameters

- **Test1**: Gets compared with Test2.
- **Test2**: Gets compared with Test1.
- **Maximum Length**: Indicates whether the number of characters is limited.
- **Ignore Case**: Indicates whether the case of the letters is taken into consideration.

Format

```
String_Not_Equal ("Test1", "Test2", Maximum Length, Ignore Case)
```

Return Value

Returns a Boolean. FALSE if Test1 and Test2 are the same string, TRUE otherwise.

Example

```
Script( String_Not_Equal_Test )
String( strEntered1 )
String( strEntered2 )
Boolean( bNotEqual )
Activate( From_Menu )
strEntered1 = Ask_String( "Type the first string", "String_Not_Equal_Test", 0, 0, "" )
strEntered2 = Ask_String( "Type the second string", "String_Not_Equal_Test", 0, 0, "" )
bNotEqual = String_Not_Equal( strEntered1, strEntered2, 0, TRUE )
If( bNotEqual )
    Message( "First string is not equal to second string", 5 )
Else
    Message( "First string is equal to second string", 5 )
End_If
Return
```
See Also

String_Empty, String_Less_Than, String_Less_Than_Or_EQUAL, String_EQUAL, String_Greater_Than_Or_EQUAL, String_Greater_Than, String_Set, Number_To_String_Decimal, Ask_String, Get_Screen_Text, Speech_To_Text, Get_Scan_Type_Name
String Only Characters

Gets a string with the specified characters. If Ignore Case is TRUE then upper-case and lower-case letters are considered to be equal.

Parameters

- **String to Parse**: The original string that gets stripped of all characters except those specified.
- **Characters to Keep**: The characters that are not stripped from the original string.
- **Ignore Case**: Indicates whether the case of the letters is taken into consideration.

Format

```
String_Only_Characters (String to Parse, Characters to Keep, Ignore Case)
```

Return Value

Returns a string where all characters in **String to Parse** that are not in **Characters to Keep** have been deleted.

Example

```
Script(String_Only_Characters_Test)
String(strEntered)
String(strResult)
String(strCharactersToKeep)
Activate(From_Menu)
    strEntered = Ask_String("Enter some text", "String Only Characters Test", 1, 99, "abcdefghijk")
    strCharactersToKeep = Ask_String("Enter the characters to keep", "String Only Characters Test", 1, 99, "abc")
    strResult = String_Only_Characters(strEntered, strCharactersToKeep, TRUE)
    Ask_OK(strResult, "String Only Characters Test")
Return
```

See Also

String Set, String Combine, String Left, String Right, String Middle, String Upper, String Lower, String Replace, String Strip Characters, String Trim Spaces Start, String Trim_
Spaces_End, Number_To_String_Binary, Number_To_String_Octal, Number_To_String_Decimal, Number_To_String_Hexadecimal_Lowercase, Number_To_String_Hexadecimal_Uppercase, Ask_String, Ask_String_Password, Ask_String_Uppercase, Ask_String_Lowercase, String_Equal
Chapter 7: Overview of Actions

String_Replace

Replaces the specified text with another string. If Ignore Case is TRUE then upper-case and lower-case letters are considered to be equal.

Parameters

- **String to Parse**: The original string that gets the substring replaced.
- **Substring to Replace**: Find all instances of this and replace them.
- **Replacement Substring**: The replacement text to use.
- **Ignore Case**: Indicates whether the case of the letters is taken into consideration.

Format

```
String_Replace ("String to Parse", "Substring to Replace", "Replacement Substring", Ignore Case)
```

Return Value

Returns a string where all instances of Substring to Replace have been replaced with Replacement Substring.

Example

```
Script( String_Replace_Test )
String( strResult )
Activate( From_Menu )
   strResult = String_Replace( "123456789012345", "2", "aaaaa", FALSE )
   Message( strResult, 0 )
Return
```

See Also

String_Set, String_Combine, String_Left, String_Right, String_Middle, String_Upper, String_Lower, String_Only_Characters, String_Strip_Characters, String_Trim_Spaces_Start, String_Trim_Spaces_End, Number_To_String_Binary, Number_To_String_Octal, Number_To_String.Decimal, Number_To_String_Hexadecimal_Lowercase, Number_To_String_Hexadecimal_Uppercase, Ask_String, Ask_String_Password, Ask_String_Uppercase, Ask_String_Lowercase, String_Equal
**String_Right**

Returns the specified characters of the input string.

### Parameters

- **String** The string from which to get the characters.
- **Sequence** The number of characters to get, starting at the beginning of the string.

### Format

```
String_Right (String, Sequence)
```

### Return Value

Returns a string with just the last \( n \) characters of the input string. If the input string is less than \( n \) characters, the entire string is returned.

### Example

```
Script(String_Right_Test)
String(strEntered)
String(strRightPart)
Activate(From_Menu)
    strEntered = Ask_String("Enter some text", "String_Right_Test", 1, 99, 
    
    strRightPart = String_Right(strEntered, 3)
Ask_OK(strRightPart, "String_Right 3 characters")
Return
```

### See Also

- [String_Set](#)
- [String_Combine](#)
- [String_Left](#)
- [String_Middle](#)
- [String_Upper](#)
- [String_Lower](#)
- [String_Replace](#)
- [String_Only_Chars](#)
- [String_Strip_Characters](#)
- [String_Trim_Spaces_Start](#)
- [String_Trim_Spaces_End](#)
- [Number_To_String_Binary](#)
- [Number_To_String_Octal](#)
- [Number_To_String.Decimal](#)
- [Number_To_String.Hexadecimal_Lowercase](#)
- [Number_To_String.Hexadecimal_Uppercase](#)
- [Ask_String](#)
- [Ask_String_Password](#)
- [Ask_String_Uppercase](#)
- [Ask_String_Lowercase](#)
- [String_Equal](#)
Chapter 7: Overview of Actions

String_Set

Assign a string value. The equal sign (=) is the symbol for String_Set in the Script Editor.

Parameters

String The String, Variable or Action that returns a string.

Format

String_Set (String)

Return Value

Returns the value of the string.

Example

Script(String_Set_Test)
String(strResult)
String(strTitle)
Activate(From_Menu)
   strTitle = "Scripting String"
   strResult = Ask_String("Enter some text", "String_Set_Test", 1, 99, ")
   strResult = String.Combine("Text Entered:", strResult)
   Ask_OK(strResult, strTitle)
Return

See Also

String_Combine, String_Left, String_Right, String_Middle, String_Upper, String_Lower, String_Replace, String_Only_Characters, String_Strip_Characters, String_Trim_Spaces_Start, String_Trim_Spaces_End, Number_To_String_Binary, Number_To_String_Octal, Number_To_String_Decimal, Number_To_String_Hexadecimal_Lowercase, Number_To_String_Hexadecimal_Uppercase, Ask_String, Ask_String_Password, Ask_String_Uppercase, Ask_String_Lowercase, String_Equal
String_Strip_Characters

Strips the specified characters from the string. If Ignore Case is TRUE then upper-case and lower-case letters are considered to be equal.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String to Parse</td>
<td>The original string that gets stripped of all characters except those specified.</td>
</tr>
<tr>
<td>Characters to Strip</td>
<td>The characters to be stripped from the original string.</td>
</tr>
<tr>
<td>Ignore Case</td>
<td>Indicates whether the case of the letters is taken into consideration.</td>
</tr>
</tbody>
</table>

Format

`String_Strip_Characters (String to Parse, Characters to Strip, Ignore Case)`

Return Value

Returns a string where all characters in `String to Parse` that are not in `Characters to Keep` have been deleted.

Example

```wolfram
Script( String_Strip_Characters_Test )
String( strEntered )
String( strResult )
String( strCharactersToStrip )
Activate( From_Menu )
    strEntered = Ask_String( "Enter some text", "String_Strip_Characters_Test", 1, 99, "abcdefghijk" )
    strCharactersToStrip = Ask_String( "Enter the characters to strip", "String_Strip_Characters_Test", 1, 99, "abc" )
    strResult = String_Strip_Characters( strEntered, strCharactersToStrip, TRUE )
    Ask_OK( strResult, "String_Strip_Characters_Test" )
Return
```

See Also

`String_Set, String_Combine, String_Left, String_Right, String_Middle, String_Upper, String_Lower, String.Replace, String_Only_Characters, String_Trim_Spaces_Start, String_Trim_...`
Spaces_End, Number_To_String_Binary, Number_To_String_Octal, Number_To_String_Decimal, Number_To_String_Hexadecimal_Lowercase, Number_To_String_Hexadecimal_Uppercase, Ask_String, Ask_String_Password, Ask_String_Uppercase, Ask_String_Lowercase, String_Equal
String_To_Number_Binary

Get a string’s binary representation. Parsing the string continues until a character other than a 0 or 1 is reached.

Parameters

String  The string that represents a number.

Format

String_To_Number_Binary (String)

Return Value

Returns the binary number represented by the string. If the string does not represent a binary number, a 0 is returned.
Example

Script( String_To_Number_Binary_Test )
String( strEntered )
String( strMessage )
String( strBinary )
String( strDecimal )
String( strHexUpper )
String( strOctal )
Number( nBinary )
Activate( From_Menu )
strEntered = Ask_String( "Enter a binary number", "String_To_Number_Binary", 1, 99, "10100101" )
nBinary = String_To_Number_Binary( strEntered )
strBinary = strEntered
strDecimal = Number_To_String_Decimal( nBinary )
strHexUpper = Number_To_String_Hexadecimal_Uppercase( nBinary )
strOctal = Number_To_String_Octal( nBinary )
strMessage = String_Combine( "Binary:", strBinary )
strMessage = String_Combine( strMessage, "; Octal: " )
strMessage = String_Combine( strMessage, strOctal )
strMessage = String_Combine( strMessage, "; Decimal: " )
strMessage = String_Combine( strMessage, strDecimal )
strMessage = String_Combine( strMessage, "; Hexadecimal:" )
strMessage = String_Combine( strMessage, strHexUpper )
Ask_OK( strMessage, "String_To_Number_Binary" )
Return

See Also

String_To_Number_Octal, String_To_Number.Decimal, String_To_Number_Hexadecimal, Number_EQUAL, String_Set, Number_Set
String_To_Number_Decimal

Gets a string’s decimal representation. Parsing the string continues until a character other than a 0 - 9 is reached.

Parameters

String  The string that represents a number.

Format

String_To_Number_Decimal (String)

Return Value

Returns the decimal (base-10) number represented by the string. If the string does not represent a decimal number, a 0 is returned.
Chapter 7: Overview of Actions

Example

Script(String_To_Number_Decimal_Test)
String(strEntered)
String(strMessage)
String(strBinary)
String(strDecimal)
String(strHexUpper)
String(strOctal)
Number(nDecimal)
Activate(From_Menu)
    strEntered = Ask_String("Enter a decimal number", "String_To_Number_Decimal", 1, 99, "45")
nDecimal = String_To_Number_Decimal(strEntered)
strDecimal = strEntered
strBinary = Number_To_String_Binary(nDecimal)
strHexUpper = Number_To_String_Hexadecimal_Uppercase(nDecimal)
strOctal = Number_To_String_Octal(nDecimal)
strMessage = String_Combine("Binary:", strBinary)
strMessage = String_Combine(strMessage, "; Octal:")
strMessage = String_Combine(strMessage, strOctal)
strMessage = String_Combine(strMessage, "; Decimal:")
strMessage = String_Combine(strMessage, strDecimal)
strMessage = String_Combine(strMessage, "; Hexadecimal:")
strMessage = String_Combine(strMessage, strHexUpper)
Ask_OK(strMessage, "String_To_Number_Decimal")
Return

See Also

String_To_Number_Binary, String_To_Number_Octal, String_To_Number_Hexadecimal,
Number_Equal, String_Set, Number_Set
String_To_Number_Hexadecimal

Gets a string’s hexadecimal representation. Parsing the string continues until a character other than a 0 - 9, a - f, or A - F is reached.

Parameters

String  The string that represents a number.

Format

String_To_Number_Hexadecimal (String)

Return Value

Returns the hexadecimal (base-16) number represented by the string. If the string does not represent a hexadecimal number, a 0 is returned.
Example

Script(String_To_Number_Hexadecimal_Test)
String(strEntered)
String(strMessage)
String(strBinary)
String(strDecimal)
String(strHexUpper)
String(strOctal)
Number(nHexadecimal)
Activate(From_Menu)
    strEntered = Ask_String("Enter a hexadecimal number", "String_To_Number_Hexadecimal", 1, 99, "A5")
    nHexadecimal = String_To_Number_Hexadecimal(strEntered)
    strHexUpper = strEntered
    strBinary = Number_To_String_Binary(nHexadecimal)
    strDecimal = Number_To_String_Decimal(nHexadecimal)
    strOctal = Number_To_String_Octal(nHexadecimal)
    strMessage = String_Combine("Binary:", strBinary)
    strMessage = String_Combine(strMessage, "; Octal:")
    strMessage = String_Combine(strMessage, strOctal)
    strMessage = String_Combine(strMessage, "; Decimal:")
    strMessage = String_Combine(strMessage, strDecimal)
    strMessage = String_Combine(strMessage, "; Hexadecimal:")
    strMessage = String_Combine(strMessage, strHexUpper)
    Ask_OK(strMessage, "String_To_Number_Hexadecimal")
Return

See Also

String_To_Number_Binary, String_To_Number_Octal, String_To_Number.Decimal, Number_Equal, String_Set, Number_Set
String_To_Number_Octal

Gets a string’s octal representation. Parsing the string continues until a character other than a 0 - 7 is reached.

Parameters

String  The string that represents a number.

Format

String_To_Number_Octal (String)

Return Value

Returns the octal (base-8) number represented by the string. If the string does not represent an octal number, a 0 is returned.
Example

```plaintext
Script(String_To_Number_Octal_Test)
String(strEntered)
String(strMessage)
String(strBinary)
String(strDecimal)
String(strHexUpper)
String(strOctal)
Number(nOctal)
Activate(From_Menu)
  strEntered = Ask_String("Enter an octal number", "String_To_Number_Octal", 1, 99, "27")
  nOctal = String_To_Number_Octal(strEntered)
  strOctal = strEntered
  strBinary = Number_To_String_Binary(nOctal)
  strDecimal = Number_To_String_Decimal(nOctal)
  strHexUpper = Number_To_String_Hexadecimal_Uppercase(nOctal)
  strMessage = String_Combine("Binary:" , strBinary)
  strMessage = String_Combine(strMessage, "; Octal:" )
  strMessage = String_Combine(strMessage, strOctal)
  strMessage = String_Combine(strMessage, "; Decimal:" )
  strMessage = String_Combine(strMessage, strDecimal)
  strMessage = String_Combine(strMessage, "; Hexadecimal:" )
  strMessage = String_Combine(strMessage, strHexUpper)
  Ask_OK(strMessage, "String_To_Number_Octal")
Return
```

See Also

`String_To_Number_Binary, String_To_Number.Decimal, String_To_Number_Hexadecimal, Number_Equal, String_Set, Number_Set`
String_Trim_Spaces_End

gets the specified text with all tabs and spaces deleted from the end.

Parameters

String to Parse  The original string which is removed of all spaces and tabs at the end.

Format

String_Trim_Spaces_End (String to Parse)

Return Value

Returns a string where all spaces and tabs at the end of the string have been deleted.

Example

Script( String_Trim_Spaces_End_Test )
String( strResult )
String( strOriginal )
String( strMessage )
Activate( From_Menu )

    strOriginal = "abcd 
    strResult = String_Trim_Spaces_End( strOriginal )
    strMessage = String_Combine( """, strOriginal )
    strMessage = String_Combine( strMessage, "" converted to "" )
    strMessage = String_Combine( strMessage, strResult )
    strMessage = String_Combine( strMessage, """" )
    Message( strMessage, 8 )
    Return

See Also

String_Set, String_Combine, String_Left, String_Right, String_Middle, String_Upper, String_Lower, String_Replace, String_Only_Characters, String_Strip_Characters, String_Trim_Spaces_Start, Number_To_String_Binary, Number_To_String_Octal, Number_To_String.Decimal, Number_To_String_Hexadecimal_Lowercase, Number_To_String_Hexadecimal_Uppercase, Ask_String, Ask_String_Password, Ask_String_Uppercase, Ask_String_Lowercase, String_Equal
String_Trim_Spaces_Start

Gets the specified text with all tabs and spaces deleted from the beginning.

Parameters

String to Parse  The original string which is removed of all spaces and tabs at the beginning.

Format

String_Trim_Spaces_Start ("String to Parse")

Return Value

Returns a string where all spaces and tabs at the start of the string have been deleted.

Example

Script( String_Trim_Spaces_Start_Test )
String( strResult )
Activate( From_Menu )
    strResult = String_Trim_Spaces_Start( "      567890" )
Message( strResult, 0 )
Return

See Also

String_Set, String_Combine, String_Left, String_Right, String_Middle, String_Upper, String_Lower, String_Replace, String_Only_Characters, String_Strip_Characters, String_Trim_Spaces_End, Number_To_String_Binary, Number_To_String_Octal, Number_To_String.Decimal, Number_To_String_Hexadecimal_Lowercase, Number_To_String_Hexadecimal_Uppercase, Ask_String, Ask_String_Password, Ask_String_Uppercase, Ask_String_Lowercase, String_Equal
Chapter 7: Overview of Actions

String_Upper

Converts the specified text to uppercase letters.

Parameters

String  The string to convert to uppercase.

Format

String_Upper (String)

Return Value

Returns a string with all characters converted to uppercase.

Example

Script(String_Upper_Lower_Test)
String(strEntered)
String(strUpper)
String(strLower)
Activate(From_Menu)
    strEntered = Ask_String("Enter some text", "String_Upper_Lower_Test", 1, 99, "")
    strUpper = String_Upper(strEntered)
    Ask_OK(strUpper, "String_Upper_Test")
    strLower = String_Lower(strEntered)
    Ask_OK(strLower, "String_Lower_Test")
Return

See Also

String_Set, String_Combine, String_Left, String_Right, String_Middle, String_Lower, String_Replace, String_Only_Characters, String_Strip_Characters, String_Trim_Spaces_Start, String_Trim_Spaces_End, Number_To_String_Binary, Number_To_String_Octal, Number_To_String_Decimal, Number_To_String_Hexadecimal_Lowercase, Number_To_String_Hexadecimal_Uppercase, Ask_String, Ask_String_Password, Ask_String_Uppercase, Ask_String_Lowercase, String_Equals
Suspend

Suspends the device.

If Prefer Hibernation is TRUE, the device will hibernate instead of suspend. (Not supported on CE devices.)

If Force Suspension is TRUE, the device will suspend immediately and other applications will not be allowed to override.

Parameters

<table>
<thead>
<tr>
<th>Prefer Hibernation</th>
<th>Indicates whether the device will hibernate instead of suspending.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Force Suspension</td>
<td>Indicates whether other applications can override the suspension.</td>
</tr>
</tbody>
</table>

Format

Suspend (Prefer Hibernation, Force Suspension)

Return Value

Returns a Boolean. TRUE if the device was suspended; returns FALSE otherwise.

Example

Script(Suspend_Test)
Boolean(bHibernate)
Boolean(bForce)
Boolean(bResult)
Activate(From_Menu)
  bHibernate = Ask_Yes_No("Hibernate instead of Suspend?", "Suspend_Test", FALSE)
  bForce = Ask_Yes_No("Force Suspend?", "Suspend_Test", FALSE)
  bResult = Suspend(bHibernate, bForce)
Return

See Also

Reboot, Get_Time_Since_Reset, Ask_Yes_No, Disconnect, Abort_All, Exit_Application
Switch

The Switch action block consists of a series of Case actions and an optional Default action, ending with an End_Switch action.

The Test parameter is compared to each of the Case parameters up until End_Switch. If Test matches the value of a Case parameter, then the actions following that Case action get executed.

If no Case parameters match Test, then the actions following Default are executed. If no Case parameters match and there is no Default action, then the actions following End_Switch get executed.

The parameter Test may be a constant or variable string or number.

Parameters

Test  Compared to each of the Case parameters up until End_Switch. If Test matches the value of a Case parameter, then the actions following that Case action get executed.

Format

Switch( Test )

Example #1

Script(Switch_Test)
String(strEntered)
Activate(From_Menu)
    strEntered = Ask_String("Type a string", "Switch_Test", 0, 0, ")
Switch( strEntered )
    Case("Hi")
        Ask_Ok("Hi", "Switch Result")
        Break
    Case("Bye")
        Ask_Ok("Bye", "Switch Result")
        Break
    Case("OK")
    Case("Ok")
    Case("ok")
        Ask_Ok("OK", "Switch Result")
        Break
Default
    Ask_Ok("Default action", "Switch Result")
    Break
End_Switch
Example #2

```plaintext
Script( switchFunctionSample )
String( strEntry )
Activate( From_Menu )
strEntry = ""
strEntry = Ask_String( "Enter a Command and Control Command:", "Help Description", 1, 20, "help" )
Switch( strEntry )
Case( "repeat" )
  Ask_OK( "Repeats a prompt.", "Repeat Command" )
Case( "speak faster" )
  Ask_OK( "Increases the cadence.", "Speak Faster Command" )
Case( "speak slower" )
  Ask_OK( "Decreases the cadence.", "Speak Slower Command" )
Case( "volume up" )
  Ask_OK( "Increases the volume.", "Increase Volume Command" )
Case( "volume down" )
  Ask_OK( "Decreases the volume.", "Decrease Volume Command" )
Case( "main menu" )
  Ask_OK( "Returns user to main menu.", "Main Menu Command" )
Case( "go back" )
  Ask_OK( "Returns user to previous screen.", "Go Back Command" )
Default
  Ask_OK( "Help commands are: repeat, speak faster, speak slower, volume up, volume down, main menu, and go back.", "Help Commands Available" )
End_Switch
Return
```

See Also

* Case, Break, Default, End_Switch*
Wait_For_Screen_Update

Suspends the current script until the screen has been updated.

Any changes to the screen will cause the script to resume; so it is recommended that you put the wait command inside a While loop, and only exit the loop once you have detected the screen you want.

Wait_For_Screen_Update may be used to wait for a script-created button to be pressed, after the action Button_Bitmap_Create_Emulation or Button_Create_Emulation.

Example

```plaintext
Script(Wait_For_Screen_Update_Test)
    Activate(From_Menu)
    Message("Waiting for screen update", 0)
    Wait_For_Screen_Update
    Message_Clear
    Return
```

See Also

Delay, Button_Bitmap_Create_Emulation, Button_Bitmap_Create_View, Button_Create_Emulation, Button_Create_View, Wait_For_Screen_Update_With_Timeout
Wait_For_Screen_Update_With_Timeout

Suspends the current script until the screen has been updated or the specified number of milliseconds have passed. Returns TRUE if the screen was updated, FALSE if the timeout occurred.

Parameters

Value1  The number of milliseconds that the script suspends.

Format

Wait_For_Screen_Update_With_Timeout(Value1)

Example

Script(Wait_For_Screen_Update_With_Timeout_Test)
    Activate(From_Menu)
    Message("Waiting a few seconds for screen update", 0)
    Wait_For_Screen_Update_With_Timeout(3000)
    Message_Clear
    Return
Web_Get_Current_Element

Returns the HTML code for the Web element with the focus. Returns an empty string if no Web element has the focus.

Format

Web_Get_Source

Example

Script( Web_Get_Current_Element_Test )
String( strCurrentElement )
Activate( From_Menu )

    strCurrentElement = Web_Get_Current_Element

Ask_OK( strCurrentElement, "Web Page Current Element Text" )
Return
Web_Get_Source

Returns the HTML code immediately following the first instance of the search string or an empty string if not found.

Returns the start of the page if Search String is blank.

If the maximum length is 0, all the data that will fit in the string is returned.

Parameters

Value1 The Search String.
Value2 The maximum number of characters to return (integer).
Value3 TRUE to ignore the letter case. FALSE if otherwise. Boolean).
Value4 TRUE to search the frames of the page. FALSE if otherwise.

Format

Web_Get_Source ("Value1", Value2, Value3, Value4)

Example

Script( Web_Get_Source_First )
String( strSource )
Activate( From_Menu )
    strSource = Web_Get_Source( ",", 0, FALSE, FALSE )
    Ask_OK( strSource, "Web Page Text" )
Return
Web_Navigate

Navigates WEB emulation to the URL provided.

Parameters

URL The URL to which the Web emulator navigates.

Format

Web_Navigate ("URL")

Return Value

Returns a Boolean. TRUE if the navigation was successful, returns FALSE otherwise.

Example

Script(Web_Navigate_to_Google)
Activate(From_Menu)
    Comment: This script when launched will navigate to www.google.com
    Web_Navigate("http://www.google.com")
    Return

See Also

Web_Navigate_Frame

Navigates WEB emulation to the URL provided within the indicated frame (Frame Name).

Parameters

URL The URL to which the Web emulator navigates.
Frame Name The name of the frame where the navigation takes place.

Format

Web_Navigate_Frame ("URL", "Frame Name")

Return Value

Returns a Boolean. TRUE if the navigation was successful, returns FALSE otherwise.

Example

Script(Web_Navigate_Frame_to_Google)
Activate(From_Menu)
Comment: This script when launched will navigate to www.google.com.
Web_Navigate_Frame("http://www.google.com",
"FrameName")
Return

See Also

Web_Navigate_Post_Data

Navigates WEB emulation to the URL provided. The Post Data is sent to the server using an HTTP POST transaction rather than an HTTP GET transaction.

Parameters

- **URL**: The URL to which the Web emulator navigates.
- **Post Data**: The string of data that the Web emulator sends to the server.

Format

Web_Navigate_Post_Data ("URL", "Post Data")

Return Value

Returns a Boolean. TRUE if the navigation was successful, returns FALSE otherwise.

Example

```plaintext
Script( Web_Navigate_Post_Data_Test )
Activate( From_Menu )
    Comment: This script when launched will post a first and last name to http://www.snee.com/xml/crud/posttest.cgi
Web_Navigate_Post_Data( "http://www.snee.com/xml/crud/posttest.cgi", "fname=MyFirstName&lname=MyLastName" )
Return
```

See Also

Web_Scripting

Instructs WEB emulation to execute the scripting information. The code should start with a javascript: or vbscript: string to ensure that the correct scripting type is used.

Parameters

**Code**  The JavaScript or VBScript that the Web emulator will execute.

Format

```web_scripting ("Code")```

Return Value

Returns a Boolean. TRUE if the script execution started successfully, returns FALSE otherwise.

Example

This example has two parts: the Wavelink script and HTML with embedded JavaScript. The following Wavelink script annunciates the item prompt and number. It also prompts for a quantity to be picked. The script then waits for a quantity to be stated. This quantity is passed back to the JavaScript.

```plaintext
Script( dnwebdemo )
String( strMessage2 )
String( strPickQty )
String( strItemPrompt2 )
String( strItem2 )
String( strCmdLn )
String( sSpeechResult )
Boolean( bSpeechStarted )
Boolean( bSpeechDone )
Number( nReadMode )

Comment: Enable Speakeasy Support
If_Not( Speech_To_Text_Available )
   Ask_OK( "Speech-to-Text is not available.", "Error" )
Return
End_If
If_Not( Speech_From_Text_Available )
   Ask_OK( "Text-to-Speech is not available.", "Error" )
Return
End_If
Speech_To_Text_Cancel```
Speech_Change_Setting( "stt_language", Speech_Find_Setting_Value( "stt_language", "enu", FALSE ) )
Speech_Change_Setting( "tts_language", Speech_Find_Setting_Value( "tts_language", "English", FALSE ) )
Comment: Ensure read mode is in sentence mode
nReadMode = Speech_Find_Setting_Value( "tts_readmode", "sentence", FALSE )
Speech_Change_Setting( "tts_readmode", nReadMode )
Comment: Annunciate Item prompt
Speech_From_Text( strItemPrompt2, TRUE )
Comment: Change read mode to character mode to annunciate part number
nReadMode = Speech_Find_Setting_Value( "tts_readmode", "character", FALSE )
Speech_Change_Setting( "tts_readmode", nReadMode )
Comment: Annunciate Item number
Speech_From_Text( strItem2, TRUE )
Comment: Change read mode back to sentence mode to annunciate task
nReadMode = Speech_Find_Setting_Value( "tts_readmode", "sentence", FALSE )
Speech_Change_Setting( "tts_readmode", nReadMode )
Comment: Annunciate task to perform
Speech_From_Text( strMessage2, TRUE )
Comment: Acquire pick quantity via speech-to-text
Comment: Initialize Speech-To-Text variables
bSpeechStarted = FALSE
bSpeechDone = FALSE
sSpeechResult = ""
Comment: Speech-To-Text Loop used to acquire Speech-To-Text
While( TRUE )
    Comment: Start Speech-To-Text if not already started
    Comment: This is needed so we start Speech_To_Text again if
    Comment: nothing was stated before it times out.
    If_Not( bSpeechStarted )
        Comment: With this Speech-To-Text function, the script
        Comment: continues script process while waiting for speech.
        bSpeechStarted = Speech_To_Text_No_Wait( bSpeechDone, sSpeechResult, "four_digits.bnf" )
    End_If
    If( bSpeechDone )
        Comment: If sSpeechResult is not empty it signifies that we
        Comment: received a speech result.
        Comment: The While loop is exited when we get a speech result.
        If_Not( String_Empty( sSpeechResult ) )
            Break
        End_If
Comment: Re-Initialize Speech engine if nothing was stated
bSpeechStarted = FALSE
bSpeechDone = FALSE
sSpeechResult = ""
Continue
End_If
Comment: Wait_For_Screen_Update waits for speech as well.
Wait_For_Screen_Update
End_While
Comment: Assign strPickQty to Speech-To-Text Result
strPickQty = sSpeechResult
Comment: Web_Scripting( "javascript:alert('sSpeechResult')" )
strCmdLn = String_Combine( "javascript:ModifyField('', strPickQty )
strCmdLn = String_Combine( strCmdLn, '');
)  
Web_Scripting( strCmdLn )  
Return

The HTML sample with embedded JavaScript uses a meta tag to call an OnLoad function. This function in turn calls the wls:dnwebdemo (Wavelink script). This script displays a prompt with an item number. It also shows an entry field where a quantity can be entered manually or through Speech-To-Text.

<html>
<head>
<Title> Simple Speakeasy WIB Demo </Title>
<!--Meta Tag used to launch OnLoad function within Wavelink Industrial Browser.-->
<META http-equiv=OnStartup content=Javascript:OnLoad();>
</head>
<script type = "text/javascript">
//Prompt at the top of the web page.
document.write("Item Number is 15469.");
// Variables used for Text-To-Speech annunciation. These values are
// passed to the Wavelink Industrial Browser script for processing.
var strItemPrompt = "Item Number is ";
var strItem = 15469;
</script>
<!--The pickqty entry field is defined in this form.-->
<form name="form1">
<input type = "text" name="entryfield" size="20" id="pickqty" value=""/>
</form>
<script type = "text/javascript">
//This message is passed to the Wavelink script so that it can be
//annunciated via Text-To-Speech.
var strMessage = "Enter quantity of items to be picked!";
//This sets the default value of the pickqty field.
var strPickQty = document.getElementById("pickqty");
strPickQty.value = "Enter quantity here.";
function ModifyField(strPickQty)
{
//Use this function to change the value of the pickqty field via Speech-
//To-Text from Wavelink Industrial Browser.
document.getElementById("pickqty").value = strPickQty;
}
//With the use of the Meta Tag defined in the Head tag above, the
OnLoad() //function is called.
function OnLoad()
{
// The variable request is set to the call of the Wavelink script. Note
// the wls prefix.
var request =
"wls:dnwe-
bdemo(strMe-
ssage2="+strMessage+",strItemPrompt2="+strItemPrompt+",strItem2="+strItem+")";
// The Wavelink script is called as a hyperlink. Wavelink Industrial
// Browser is required for this call to be understood.
window.location.href = request;
}
</script>
<body>
</body>
</html>

See Also

# Web_Search_Source

Searches the page source of the current WEB emulation page. If \Search Frames\ is TRUE, the page source of any frames will be searched as well.

## Parameters

- **Search Text**: The text to search for in the page source.
- **Ignore Case**: Indicates whether the case of the letters is taken into consideration.
- **Search Frames**: Indicates whether the page source for the frames is searched as well.

## Format

```
Web_Search_Source ("Search Text", Ignore Case, Search Frames)
```

## Return Value

Returns a Boolean. TRUE if the text is found anywhere in the page source, returns FALSE otherwise.

## Example

```
Script(Web_Search_Source_Test)
Boolean(bFound)
Activate(From_Menu)
    bFound = Web_Search_Source("wavelink", TRUE, TRUE)
    If(bFound)
        Message("I found the word WAVELINK", 5)
    End_If
Return
```

## See Also

While

If the test is TRUE, the statements after While and before the next EndWhile statement are executed and the While statement will be executed again. Otherwise, execution will proceed to the next EndWhile statement.

The While loop will continue to execute until the test fails, a Break command is executed, or the script exits.

Parameters

Test As long as this is TRUE, then the actions up to the End While get executed.

Format

While (Test)

Example

Script(While_Test)
    Boolean(bOK)
    Activate(From_Menu)
    bOK = TRUE
    While(bOK)
        bOK = Ask_OK_Cancel("Press OK to keep getting this message.", "Test", FALSE)
    End_While
Return

See Also

While_Not, End_While, Continue, Break
While_Not

If the test is FALSE, the statements after While and before the next EndWhile statement are executed and the While statement will be executed again. Otherwise, execution will proceed to the next EndWhile statement.

The While loop will continue to execute until the test succeeds, a Break command is executed, or the script exits.

Parameters

Test As long as this is FALSE, then the actions up to the End_While get executed.

Format

While_Not (Test)

Example

Script( While_Not_Test )
  Boolean( bOK )
  Activate( From_Menu )
    bOK = FALSE
    While_Not( bOK )
      bOK = Ask_OK_Cancel( "Press Cancel to keep getting this message.", "Test", FALSE )
  End_While
  Return

See Also

While, End_While, Continue, Break
Chapter 8: Speakeasy Settings

This section lists settings supported by Speakeasy. These settings are to be used in conjunction with the preceding scripting actions.

Wavelink recommends grouping Speakeasy settings together at the beginning of a script.

**Text-to-Speech Settings**

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tts_calibrate</td>
<td>Opens the speaker volume calibration wizard.</td>
</tr>
<tr>
<td>tts_external_speaker_setting</td>
<td>Speaker setting for use on Motorola/Symbol mobile devices.</td>
</tr>
<tr>
<td>tts_frequency</td>
<td>Indicates the sampling frequency.</td>
</tr>
<tr>
<td>tts_language_long</td>
<td>Displays the full name of the language currently being used.</td>
</tr>
<tr>
<td>tts_language_short</td>
<td>Displays the three-letter abbreviation of the language currently being used.</td>
</tr>
<tr>
<td>tts_pitch</td>
<td>Indicates the pitch level of spoken text.</td>
</tr>
<tr>
<td>tts_rate</td>
<td>Indicates the speed level.</td>
</tr>
<tr>
<td>tts_readmode</td>
<td>Indicates how text should be separated.</td>
</tr>
<tr>
<td>tts_voice</td>
<td>Indicates the name of the voice that is currently selected.</td>
</tr>
<tr>
<td>tts_volume</td>
<td>Indicates the sound level.</td>
</tr>
<tr>
<td>tts_waitfactor</td>
<td>Indicates the length of the pause between messages.</td>
</tr>
</tbody>
</table>
## Speech-to-Text Settings

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stt_accuracy</td>
<td>This value affects the trade-off between CPU load, memory requirements, and accuracy.</td>
</tr>
<tr>
<td>stt_adjust_gain</td>
<td>This feature allows the engine to automatically increase and decrease the microphone input volume.</td>
</tr>
<tr>
<td>stt_beep_threshold</td>
<td>If the confidence value for a result is below this value, then a negative acknowledgement beep will not be played.</td>
</tr>
<tr>
<td>stt_calibrate</td>
<td>Opens the microphone calibration wizard.</td>
</tr>
<tr>
<td>stt_calibration_silence</td>
<td>Sets how long the user is expected to remain silent during a quick microphone calibration.</td>
</tr>
<tr>
<td>stt_confidence</td>
<td>Indicates the minimum difference in confidence required between the top two speech-to-text results for the top result to be accepted.</td>
</tr>
<tr>
<td>stt_expanded</td>
<td>Use this to get the confidence value along with the speech-to-text result.</td>
</tr>
<tr>
<td>stt_fx_detect_start</td>
<td>Indicates the action the speech engine should take before attempting to determine what the user is saying.</td>
</tr>
<tr>
<td>stt_fx_microphone</td>
<td>Tells the speech engine the distance between the user and the microphone.</td>
</tr>
<tr>
<td>stt_fx_min_duration</td>
<td>Indicates the minimum duration (in ms) of speech before speech detection is activated.</td>
</tr>
<tr>
<td>stt_fx_sensitivity</td>
<td>Indicates the speech detection sensitivity.</td>
</tr>
<tr>
<td>stt_fx_silence</td>
<td>Indicates the milliseconds of silence used to indicate the user is done speaking.</td>
</tr>
<tr>
<td>stt_fx_threshold</td>
<td>Indicates the amount of energy the microphone input must have before the speech detection is activated.</td>
</tr>
<tr>
<td>Setting</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>stt_idle_timeout</td>
<td>Indicates the total milliseconds for the engine to continue collecting results following the last result or timeout.</td>
</tr>
<tr>
<td>stt_language_long</td>
<td>Displays the full name of the language currently being used.</td>
</tr>
<tr>
<td>stt_language_short</td>
<td>Displays the three-letter abbreviation of the language currently being used.</td>
</tr>
<tr>
<td>stt_logging</td>
<td>Creates a Speech-to-Text log file in the root folder of the device.</td>
</tr>
<tr>
<td>stt_logging_audio</td>
<td>Sets the engine to log speech-to-text attempts as .wav files.</td>
</tr>
<tr>
<td>stt_logging_engine</td>
<td>If set to 1, the speech-to-text engine will create a log file in the root folder of the device.</td>
</tr>
<tr>
<td>stt_pool_size</td>
<td>Sets the number of terms the engine will examine closely for the best match.</td>
</tr>
<tr>
<td>stt_preserve</td>
<td>Causes the speech engine to save the current engine state for use later.</td>
</tr>
<tr>
<td>stt_priority</td>
<td>Determines how aggressively the microphone input is collected and speech analysis is performed.</td>
</tr>
<tr>
<td>stt_processing</td>
<td>Indicates the action the speech engine should take when returning a grammar result.</td>
</tr>
<tr>
<td>stt_reset</td>
<td>Modifies engine adaptation speed and/or saved engine information.</td>
</tr>
<tr>
<td>stt_reset_session_delay</td>
<td>Indicates the total milliseconds for the speech engine to wait for a valid response before reverting back to the last saved state.</td>
</tr>
<tr>
<td>stt_result_sound</td>
<td>Causes a sound to play for result recognition.</td>
</tr>
<tr>
<td>stt_save_increase</td>
<td>Increases the threshold for saving a new engine state as time progresses.</td>
</tr>
<tr>
<td>Setting</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>stt_save_session_delay</td>
<td>Indicates the total milliseconds for the speech engine to wait before saving the next current state.</td>
</tr>
<tr>
<td>stt_save_threshold</td>
<td>Directs the speech engine to save the state if the result confidence is greater than the result confidence for stt_threshold and stt_save_threshold combined.</td>
</tr>
<tr>
<td>stt_server_timeout</td>
<td>When uploading or downloading user training data, the value for this setting is how long (in seconds) the Client will wait for a response from the Avalanche server.</td>
</tr>
<tr>
<td>stt_size</td>
<td>Displays the size of the speech-to-text engine being used.</td>
</tr>
<tr>
<td>stt_special_sounds</td>
<td>Indicates how the speech engine should interpret special sounds.</td>
</tr>
<tr>
<td>stt_threshold</td>
<td>Indicates the minimum amount of confidence for the most-likely result that will be accepted.</td>
</tr>
<tr>
<td>stt_timeout</td>
<td>Indicates the total milliseconds (ms) for the system to wait before responding to the speaker.</td>
</tr>
<tr>
<td>stt_use_jumpback</td>
<td>Sets a buffer to check if the engine is processing speech.</td>
</tr>
<tr>
<td>stt_use_word_ids</td>
<td>Enables support for Word IDs (the !id directive) in grammar files.</td>
</tr>
<tr>
<td>stt_volume</td>
<td>Indicates the current volume of the microphone input.</td>
</tr>
</tbody>
</table>
**tts_calibrate**

Opens the speaker volume calibration wizard when this is set to 0.

---

**Example**

```plaintext
Script(Alt_F10_Speaker_Calibrate)
Activate(On_Key, 0x79, Alt)
Comment: Pressing Alt-F10 displays the speaker-calibration dialog.
Speech_Change_Setting("tts_calibrate", 0)
Return
```

**See Also**

- `Speech_Get_Setting`
- `Speech_Get_Setting_Value_Desc`
- `Speech_Get_Setting_Max`
- `Speech_Find_Setting_Value`
- `Speech_To_Text`
- `Speech_From_Text`
- `Speech_Setting_Available`
- `Speech_Change_Setting`
tts_externalSpeaker_setting

Speaker setting for use on Motorola/Symbol mobile devices. If set to 1, only the CE device master volume will be adjusted. If set to 0, the volume will apply to the headset volume.

This setting is ignored for mobile devices from manufacturers other than Motorola/Symbol.

Possible Values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>on</td>
</tr>
<tr>
<td>0</td>
<td>off</td>
</tr>
</tbody>
</table>

The default value is 0 (off).

Example

```script
Script( Use_External_Speaker )
Activate( From_Menu )
    Comment: Change the speaker setting so Symbol devices volume changes affect the external speaker.
    If_Not( Speech_Change_Setting( "tts_externalSpeaker", 1 ) )
        Ask_OK( "This setting is not supported. Update your Vocalizer version.", "Error" )
    Return
    End_If
    If_Not( Number_EQUAL( Speech_Get_Setting( "tts_externalSpeaker" ), 1 ) )
    Ask_OK( "The setting change was not preserved!", "Error" )
    Return
End_If
Comment: Perform a calibration so the user can set the volume.
Speech_Change_Setting( "tts_calibrate", 0 )
Return
```

See Also

tts_volume, Speech_Get_Setting, Speech_Get_Setting_Value_Desc, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_To_Text, Speech_From_Text, Speech_Setting_Available, Speech_Change_Setting
tts_frequency

Indicates the sampling frequency.

Possible Values

- 8KHz
- 11KHz
- 16KHz
- 22KHz

Example

```plaintext
Script( Speech_From_Text_Frequency )
String( strDescription )
String( strSetting )
String( strMessage )
Number( nFrequency )
Activate( From_Menu )
    If_Not( Speech_From_Text_Available )
        Message( "Speech From Text Not Available", 3 )
        Return
    End_If
strSetting = "tts_frequency"
nFrequency = Speech_Get_Setting( strSetting )
strDescription = Speech_Get_Setting_Value_Desc( strSetting, nFrequency )
strMessage = String_Combine( "Sampling Frequency:", strDescription )
strMessage = String_Combine( strMessage, "kilohertz; setting value:" )
strMessage = String_Combine( strMessage, Number_To_String_Deimal( nFrequency ) )
    Speech_From_Text( strMessage, FALSE )
    Return
```

See Also

- Speech_Get_Setting
- Speech_Get_Setting_Value_Desc
- Speech_Get_Setting_Max
- Speech_Find_Setting_Value
- Speech_To_Text
- Speech_From_Text
- Speech_Setting_Available
- Speech_Change_Setting
tts_language_long

Displays the full name of the language currently being used.

For possible language values, see tts_language_short on page 318.

**NOTE:** An acceptable substitute for tts_language_long is tts_language, but tts_language_long is preferred.

Example

Script(Speech_Languages_Voices_Test)
Number(nVoice)
Number(nLanguage)
Activate(From_Menu)
    If_Not(Speech_From_Text_Available)
        Message("Speech From Text Not Available", 3)
        Return
    End_If
nLanguage = Speech_Find_Setting_Value("tts_language_long", "Mexican Spanish", FALSE)
    If(Number_Greater_Than_Or_Equal(nLanguage, 0))
        Speech_Change_Setting("tts_language_long", nLanguage)
    End_If
nVoice = Speech_Find_Setting_Value("tts_voice", "Javier", FALSE)
    If(Number_Greater_Than_Or_Equal(nVoice, 0))
        Speech_Change_Setting("tts_voice", nVoice)
        Speech_From_Text("La voc de Javier esta disponible.", FALSE)
    End_If
nVoice = Speech_Find_Setting_Value("tts_voice", "Paulina", FALSE)
    If(Number_Greater_Than_Or_Equal(nVoice, 0))
        Speech_Change_Setting("tts_voice", nVoice)
        Speech_From_Text("La voz de Paulina esta disponible.", FALSE)
    End_If
nLanguage = Speech_Find_Setting_Value("tts_language_long", "English USA", FALSE)
    If(Number_Greater_Than_Or_Equal(nLanguage, 0))
        Speech_Change_Setting("tts_language_long", nLanguage)
    End_If
nVoice = Speech_Find_Setting_Value("tts_voice", "tom", FALSE)
    If(Number_Greater_Than_Or_Equal(nVoice, 0))
        Speech_Change_Setting("tts_voice", nVoice)
        Speech_From_Text("Tom’s voice is available.", FALSE)
    End_If
nVoice = Speech_Find_Setting_Value("tts_voice", "samantha", FALSE)
If(Number_Greater_Than_Or_Equal(nVoice, 0))
    Speech_Change_Setting("tts_voice", nVoice)
    Speech_From_Text("Samantha’s voice is available.", FALSE)
End If
nVoice = Speech_Find_Setting_Value("tts_voice", "jill", FALSE)
If(Number_Greater_Than_Or_Equal(nVoice, 0))
    Speech_Change_Setting("tts_voice", nVoice)
    Speech_From_Text("Jill’s voice is available.", FALSE)
End If
Message("Speech Voice Testing Completed.", 3)
Return

See Also

tts_language_short, Speech_Get_Setting, Speech_Get_Setting_Value_Desc, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_To_Text, Speech_From_Text, Speech_Setting_Available, Speech_Change_Setting
### tts_language_short

Displays the three-letter abbreviation of the language currently being used. For example, ENU instead of English USA.

Possible values may include:

<table>
<thead>
<tr>
<th>Code</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARE</td>
<td>Arabic Egypt</td>
</tr>
<tr>
<td>ARS</td>
<td>Arabic Saudi Arabia</td>
</tr>
<tr>
<td>ARA</td>
<td>Arabic United Arab Emirates</td>
</tr>
<tr>
<td>ARW</td>
<td>Arabic Worldwide</td>
</tr>
<tr>
<td>AMA</td>
<td>Armenian Armenia</td>
</tr>
<tr>
<td>BAE</td>
<td>Basque Spain</td>
</tr>
<tr>
<td>BEI</td>
<td>Bengali India</td>
</tr>
<tr>
<td>BGB</td>
<td>Bulgarian Bulgaria</td>
</tr>
<tr>
<td>CAH</td>
<td>Cantonese Hong Kong</td>
</tr>
<tr>
<td>CAA</td>
<td>Catalan Andorra</td>
</tr>
<tr>
<td>CAE</td>
<td>Catalan Spain</td>
</tr>
<tr>
<td>HRH</td>
<td>Croatian Croatia</td>
</tr>
<tr>
<td>CZC</td>
<td>Czech Czech Republic</td>
</tr>
<tr>
<td>DAD</td>
<td>Danish Denmark</td>
</tr>
<tr>
<td>DUB</td>
<td>Dutch Belgium</td>
</tr>
<tr>
<td>DUN</td>
<td>Dutch The Netherlands</td>
</tr>
<tr>
<td>ENA</td>
<td>English Australia</td>
</tr>
<tr>
<td>ENC</td>
<td>English Canada</td>
</tr>
<tr>
<td>ENI</td>
<td>English India</td>
</tr>
<tr>
<td>ENE</td>
<td>English Ireland</td>
</tr>
<tr>
<td>ENN</td>
<td>English New Zealand</td>
</tr>
<tr>
<td>Code</td>
<td>Language</td>
</tr>
<tr>
<td>------</td>
<td>--------------</td>
</tr>
<tr>
<td>ENS</td>
<td>English</td>
</tr>
<tr>
<td>ENZ</td>
<td>English</td>
</tr>
<tr>
<td>ENG</td>
<td>English</td>
</tr>
<tr>
<td>ENU</td>
<td>English</td>
</tr>
<tr>
<td>FIF</td>
<td>Finnish</td>
</tr>
<tr>
<td>FRB</td>
<td>French</td>
</tr>
<tr>
<td>FRC</td>
<td>French</td>
</tr>
<tr>
<td>FRF</td>
<td>French</td>
</tr>
<tr>
<td>FRL</td>
<td>French</td>
</tr>
<tr>
<td>FNC</td>
<td>French</td>
</tr>
<tr>
<td>GEA</td>
<td>German</td>
</tr>
<tr>
<td>GEB</td>
<td>German</td>
</tr>
<tr>
<td>GED</td>
<td>German</td>
</tr>
<tr>
<td>GRL</td>
<td>German</td>
</tr>
<tr>
<td>GEL</td>
<td>German</td>
</tr>
<tr>
<td>GEC</td>
<td>German</td>
</tr>
<tr>
<td>GRG</td>
<td>Greek</td>
</tr>
<tr>
<td>HAH</td>
<td>Haitian</td>
</tr>
<tr>
<td>HEI</td>
<td>Hebrew</td>
</tr>
<tr>
<td>HII</td>
<td>Hindi</td>
</tr>
<tr>
<td>HUUH</td>
<td>Hungarian</td>
</tr>
<tr>
<td>ISI</td>
<td>Icelandic</td>
</tr>
<tr>
<td>IDI</td>
<td>Indonesian</td>
</tr>
<tr>
<td>IRE</td>
<td>Irish</td>
</tr>
<tr>
<td>ITI</td>
<td>Italian</td>
</tr>
<tr>
<td>ITC</td>
<td>Italian</td>
</tr>
<tr>
<td>Code</td>
<td>Language</td>
</tr>
<tr>
<td>------</td>
<td>------------</td>
</tr>
<tr>
<td>JPJ</td>
<td>Japanese</td>
</tr>
<tr>
<td>KAI</td>
<td>Kannada</td>
</tr>
<tr>
<td>KSI</td>
<td>Kashmiri</td>
</tr>
<tr>
<td>KOK</td>
<td>Korean</td>
</tr>
<tr>
<td>MLI</td>
<td>Malayalam</td>
</tr>
<tr>
<td>MNC</td>
<td>Mandarin</td>
</tr>
<tr>
<td>MNH</td>
<td>Mandarin</td>
</tr>
<tr>
<td>MNT</td>
<td>Mandarin</td>
</tr>
<tr>
<td>MAI</td>
<td>Marathi</td>
</tr>
<tr>
<td>NON</td>
<td>Norwegian</td>
</tr>
<tr>
<td>NNN</td>
<td>Norwegian</td>
</tr>
<tr>
<td>PLP</td>
<td>Polish</td>
</tr>
<tr>
<td>PTB</td>
<td>Portuguese</td>
</tr>
<tr>
<td>PTP</td>
<td>Portuguese</td>
</tr>
<tr>
<td>ROR</td>
<td>Romanian</td>
</tr>
<tr>
<td>RUR</td>
<td>Russian</td>
</tr>
<tr>
<td>SKS</td>
<td>Slovak</td>
</tr>
<tr>
<td>SLS</td>
<td>Slovenian</td>
</tr>
<tr>
<td>SPM</td>
<td>Spanish</td>
</tr>
<tr>
<td>SPE</td>
<td>Spanish</td>
</tr>
<tr>
<td>SPU</td>
<td>Spanish</td>
</tr>
<tr>
<td>SWS</td>
<td>Swedish</td>
</tr>
<tr>
<td>THT</td>
<td>Thai</td>
</tr>
</tbody>
</table>
### Example

```csharp
Script(Speech_From_Text_Settings_Language)
String(strDescription)
String(strSetting)
Number(nLanguage)
Activate(From_Menu)
    If_Not(Speech_From_Text_Available)
        Message("Speech From Text Not Available", 3)
    Return
End_If
strSetting = "tts_language_short"
nLanguage = Speech_Get_Setting(strSetting)
strDescription = Speech_Get_Setting_Value_Desc(strSetting, nLanguage)
Ask_OK(strDescription, strSetting)
strSetting = "tts_language_long"
nLanguage = Speech_Get_Setting(strSetting)
strDescription = Speech_Get_Setting_Value_Desc(strSetting, nLanguage)
Ask_OK(strDescription, strSetting)
strSetting = "tts_language"
nLanguage = Speech_Get_Setting(strSetting)
strDescription = Speech_Get_Setting_Value_Desc(strSetting, nLanguage)
Ask_OK(strDescription, strSetting)
Return
```

### See Also

`tts_language_long`, `Speech_Get_Setting`, `Speech_Get_Setting_Value_Desc`, `Speech_Get_Setting_Max`, `Speech_Find_Setting_Value`, `Speech_To_Text`, `Speech_From_Text`, `Speech_Setting_Available`, `Speech_Change_Setting`
**tts_pitch**

Indicates the pitch level of spoken text.

**Possible Values**

Any number from 1-100, with 50 being the default (and normal pitch). Values below 50 are lower-pitched; values above 50 are higher-pitched.

**Example**

```plaintext
Script( Pitch_Values )
Number( nValue )
Activate( From_Menu )
    nValue = Speech_Get_Setting( "tts_pitch" )
    Message( String_Combine( "Initial pitch is ", Number_To_String_Decimal( nValue ) ), 5 )
    Speech_From_Text( "This is the initial pitch.", TRUE )
    Speech_Change_Setting( "tts_pitch", 10 )
    Speech_From_Text( "This is a low pitch.", TRUE )
    Speech_Change_Setting( "tts_pitch", 100 )
    Speech_From_Text( "This is a high pitch.", TRUE )
    Speech_Change_Setting( "tts_pitch", nValue )
Return
```

**See Also**

Speech_Get_Setting, Speech_Get_Setting_Value_Desc, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_To_Text, Speech_From_Text, Speech_Setting_Available, Speech_Change_Setting
**tts_rate**

Indicates how fast the text should be spoken. Wavelink recommends using the calibration tool to adjust this setting, rather than setting it through a script or the screen reformatter.

**Possible Values**

Any number from 50-400, with 100 being the default (and "normal" talking speed), 50 being 1/2 normal speed, and 400 being 4 times faster than normal speed.

**Example #1**

This script is designed to be triggered by a speech-to-text global action to increase the speech rate. The term “faster” could be added to a grammar file to act as a command that calls the following script.

```wavelink
Script( IncreaseSpeechRate )
String( sRate )
String( sMessage )
Number( nRate )

Comment: Increase Speech rate by 10 increments
nRate = Speech_Get_Setting( "tts_rate" )
If( Number_Equal( nRate, 400 ) )
   Speech_From_Text( "Speech Rate already at maximum." , TRUE )
   Message( "Speech Rate already at maximum." , 5 )
Else
   nRate = Number_Plus( nRate, 10 )
   Speech_Change_Setting( "tts_rate", nRate )
   sRate = Number_To_String_Decimal( nRate )
   sMessage = "New Speech Rate is: "
   sMessage = String_Combine( sMessage, sRate )
   Message( sMessage, 5 )
End_If
Return
```

**Example #2**

This script is designed to be triggered by a speech-to-text global action to decrease the speech rate. The term "slower" could be added to a grammar file to act as a command that calls the following script.

```wavelink
Script( DecreaseSpeechRate )
String( sRate )
String( sMessage )
Number( nRate )
```
Comment: Decrease Speech rate by 10 increments
nRate = Speech_Get_Setting( "tts_rate" )
If( Number_Equal( nRate, 50 ) )
    Speech_From_Text( "Speech Rate already at minimum.", TRUE )
    Message( "Speech Rate already at minimum.", 5 )
Else
    nRate = Number_Minus( nRate, 10 )
    Speech_Change_Setting( "tts_rate", nRate )
    sRate = Number_To_String.Decimal( nRate )
    sMessage = "New Speech Rate is: "
    sMessage = String_Combine( sMessage, sRate )
    Message( sMessage, 5 )
End_If
Return

Example #3

Script(Speech_Rates)
Number(nRate)
Activate(From_Menu)
nRate = Speech_Get_Setting("tts_rate")
Message(String_Combine("Initial speech rate is ",
    Number_To_String.Decimal(nRate)), 5)
Speech_From_Text("This is the current speed.", TRUE)
Speech_Change_Setting("tts_rate", 10)
Speech_From_Text("This is the slow text.", TRUE)
Speech_Change_Setting("tts_rate", 99)
Speech_From_Text("This is the fast text.", TRUE)
Speech_Change_Setting("tts_rate_", nRate)
Return

See Also

Speech_Get_Setting, Speech_Get_Setting_Value_Desc, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_To_Text, Speech_From_Text, Speech_Setting_Available, Speech_Change_Setting
**tts_readmode**

Indicates how text should be separated.

**Possible Values**

- **Sentence**: The vocalizer will pronounce the words with attention to punctuation.
- **Character**: The vocalizer will pronounce each character.
- **Word**: The vocalizer will pronounce each word.
- **Character Case**: The vocalizer will say "capital" before each upper-case letter is pronounced.

**Example**

```plaintext
Script(Speech_From_Text_ReadMode)
String(strDescription)
String(strSetting)
String(strMessage)
Number(nReadMode)
Activate(From_Menu)
    If_Not(Speech_From_Text_Available)
        Message("Speech From Text Not Available", 3)
        Return
    End_If
strSetting = "tts_readmode"
nReadMode = Speech_Get_Setting(strSetting)
strDescription = Speech_Get_Setting_Value_Desc(strSetting, nReadMode)
strMessage = String_Combine("Read-mode:", strDescription)
strMessage = String_Combine(strMessage, "; setting value:")
strMessage = String_Combine(strMessage, Number_To_String_Decimal(nReadMode))
Speech_From_Text(strMessage, FALSE)
Return
```

**See Also**

Speech_Get_Setting, Speech_Get_Setting_Value_Desc, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_To_Text, Speech_From_Text, Speech_Setting_Available, Speech_Change_Setting
**tts_voice**

Indicates the name of the voice that is currently selected.

**Example**

```plaintext
Script(Jill_Voice)
Number(nVoice)
Activate(From_Menu)
    If_Not(Speech_From_Text_Available)
        Message("Speech From Text Not Available", 3)
        Return
    End_If
    nVoice = Speech_Find_Setting_Value("tts_voice", "Jill", FALSE)
    If(Number_Greater_Than_Or_Equal(nVoice, 0))
        Speech_Change_Setting("tts_voice", nVoice)
        Speech_From_Text("Jill’s voice is available.", TRUE)
    Else
        Speech_From_Text("Jill’s voice is not available.", TRUE)
    End_If
    Return

See Also

Speech_Get_Setting, Speech_Get_Setting_Value_Desc, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_To_Text, Speech_From_Text, Speech_Setting_Available, Speech_Change_Setting
```
tts_volume

Indicates the sound level. Wavelink recommends using the calibration tool to adjust this setting, rather than setting it through a script or the screen reformatter.

Possible Values

Any number from 0 to 100, with the default being the value last set in the calibration tool. For devices that support "boost" (a hardware amplifier that can be enabled or disabled), values of 51-100 will be boosted, while 0-50 will not. For devices that don't support boost, the range of 0-100 will just be softest to loudest.

Example #1

This script is designed to be triggered by a speech-to-text global action to increase the volume. The word "louder" could be added to a grammar file to act as a command that calls the following script.

```plaintext
Script( IncreaseVolume )
String( sVolume )
String( sMessage )
Number( nVolume )
Number( nVolumeCalc )

Comment: Increase Volume by 10%

nVolume = Speech_Get_Setting( "tts_volume" )
nVolumeCalc = Number_Multiply( nVolume, 11 )
nVolume = Number_Divide( nVolumeCalc, 10 )
If( Number_Greater_Divide( nVolume, 100 ) )
   Speech_From_Text( "Volume already at maximum.", TRUE )
   Message( "Volume already at maximum.", 5 )
   Speech_Change_Setting( "tts_volume", 100 )
Else
   Speech_Change_Setting( "tts_volume", nVolume )
   sVolume = Number_To_String_Decimal( nVolume )
   sMessage = "New Volume Level is: "
   sMessage = String_Combine( sMessage, sVolume )
   Message( sMessage, 5 )
End_If
Return
```

Example #2

This script is designed to be triggered by a speech-to-text global action to decrease the volume. The term "softer" could be added to a grammar file to act as a command that calls the following
Script.

Script( DecreaseVolume )
String( sVolume )
String( sMessage )
Number( nVolume )
Number( nVolumeCalc )

Comment: Decrease Volume by 10%

nVolume = Speech_Get_Setting( "tts_volume" )
nVolumeCalc = Number_Multiply( nVolume, 9 )
nVolume = Number_Divide( nVolumeCalc, 10 )

If( Number_Less_Than_Or_Equal( nVolume, 10 ) )
    Speech_From_Text( "Volume already at minimum.", TRUE )
    Message( "Volume already at minimum.", 5 )
    Speech_Change_Setting( "tts_volume", 10 )
Else
    Speech_Change_Setting( "tts_volume", nVolume )
    sVolume = Number_To_String_Decimal( nVolume )
    sMessage = "New Volume Level is:"
    sMessage = String_Combine( sMessage, sVolume )
    Message( sMessage, 5 )
End_If
Return

Example #3

Script( Speech_From_Text_Volume )
String( strDescription )
String( strSetting )
String( strMessage )
String( strPrompt )
Boolean( bResult )
Number( nVolume )
Number( nSettingMax )
Activate( From_Menu )

If_Not( Speech_From_Text_Available )
    Message( "Speech From Text Not Available", 3 )
    Return
End_If
strSetting = "tts_volume"
nVolume = Speech_Get_Setting( strSetting )
strDescription = Speech_Get_Setting_Value_Desc( strSetting, nVolume )
strMessage = String_Combine( "Volume: ", strDescription )
strMessage = String_Combine( strMessage, "; setting value: " )
strMessage = String_Combine( strMessage, Number_To_String_Decimal(
nVolume )
    Speech_From_Text( strMessage, FALSE )

nSettingMax = Speech_Get_Setting_Max( strSetting )
strPrompt = String_Combine( "From 0 to ", Number_To_String_Decimal( nSettingMax ) )

While( Number_Not_EQUAL( nVolume, 0 ) )
    nVolume = Ask_Number( strPrompt, "New volume, 0 to exit", 0, nSettingMax, nVolume )
    If( Number_Not_EQUAL( nVolume, 0 ) )
        bResult = Speech_Change_Setting( strSetting, nVolume )
        If( bResult )
            strMessage = String_Combine( "The new volume level is " String_Decimal( nVolume ) )
            Speech_From_Text( strMessage, FALSE )
        Else
            Message( "Setting volume failed", 3 )
            nVolume = 0
    End_If
End_While
Return

See Also

stt_volume, Speech_Get_Setting, Speech_Get_Setting_Value_Desc, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_To_Text, Speech_From_Text, Speech_Setting_Available, Speech_Change_Setting
**tts_waitfactor**

Indicates the length of the pause between messages.

---

**Possible Values**

- 0 milliseconds (ms)
- 200 ms
- 400 ms
- 600 ms
- 800 ms
- 1000 ms
- 1200 ms
- 1400 ms
- 1600 ms
- 1800 ms

---

**Example**

```plaintext
Script( Speech_From_Text_WaitFactor )
String( strDescription )
String( strSetting )
String( strMessage )
Number( nWaitFactor )
Activate( From_Menu )
    If_Not( Speech_From_Text_Available )
        Message( "Speech From Text Not Available", 3 )
        Return
    End_If
strSetting = "tts_waitfactor"
nWaitFactor = Speech_Get_Setting( strSetting )
strDescription = Speech_Get_Setting_Value_Desc( strSetting, nWaitFactor )
    strMessage = String_Combine( "Wait-factor:", strDescription )
    strMessage = String_Combine( strMessage, ";milliseconds; setting value:" )
    strMessage = String_Combine( strMessage, Number_To_String.Decimal( nWaitFactor ) )
    Speech_From_Text( strMessage, FALSE )
Return
```
See Also

Speech_Get_Setting, Speech_Get_Setting_Value_Desc, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_To_Text, Speech_From_Text, Speech_Setting_Available, Speech_Choose_Setting
**stt_accuracy**

This value affects the trade-off between CPU load, memory requirements, and accuracy. With a lower value, less CPU time and memory are needed, but the accuracy also degrades. With a higher value, more CPU time and memory are needed, but performance increases.

**Possible Values**

1-1000. The default value is 100 for mobile devices, and 200 for desktop or laptop computers.
stt_adjust_gain

This setting allows the engine to automatically increase and decrease the microphone input volume. This is disabled by default. This feature will often cause the voice detection rate to deteriorate, so its use is not recommended.

Possible Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>disabled</td>
</tr>
<tr>
<td>1</td>
<td>enabled</td>
</tr>
<tr>
<td>2</td>
<td>enabled; save as the new default volume</td>
</tr>
</tbody>
</table>

Example

```
Script( Speech_Adjust_Gain )
Activate( From_Menu )
    If( Ask_Yes_No( "Is microphone calibration just not good enough?", "Gain Option", FALSE ) )
        Speech_Change_Setting( "stt_adjust_gain", 1 )
        Message( "The volume will be adjusted automatically.", 5 )
    Else
        Speech_Change_Setting( "stt_adjust_gain", 0 )
        Message( "The volume will not be changed.", 5 )
    End_If
Return
```
**stt_beep_threshold**

If the confidence value for a result is below this value, then a negative acknowledgement beep will not be played. The default value is 3000. Set the value to 0 to disable this feature.

**Possible Values**

0-10000. When the value is set to 0, the feature is disabled.
**stt_calibrate**

Opens the microphone calibration wizard.

**Possible Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>starts the calibration wizard</td>
</tr>
<tr>
<td>1</td>
<td>performs a quick calibration</td>
</tr>
<tr>
<td>2</td>
<td>launches the user training</td>
</tr>
</tbody>
</table>

**Example**

```plaintext
Script( Speech_Calibration )
Activate( From_Menu )
    If_Not( Speech_From_Text_Available )
        Comment: Only the calibration wizard is available.
        Speech_Change_Setting( "stt_calibrate", 0 )
    Return
End_If

    If( Ask_Yes_No( "Do you want to perform user training?", "User Training", TRUE ) )
        Speech_Change_Setting( "stt_calibrate", 2 )
    Return
End_If

    If( Ask_Yes_No( "Do you want to perform a full calibration?", "Full Calibration", TRUE ) )
        Speech_Change_Setting( "stt_calibrate", 0 )
    Else
        Speech_Change_Setting( "stt_calibrate", 1 )
    End_If
Return
```

**See Also**

* stt_volume, stt_fx_microphone, Speech_Get_Setting, Speech_Get_Setting_Value_Desc, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_To_Text, Speech_From_Text, Speech_Setting_Available, Speech_Change_Setting
**stt_calibration_silence**

Sets how long the user is expected to remain silent during a quick microphone calibration.

**Possible Values**

Any value from 0-60000 ms. The default is 5000 ms (5 seconds). If the value is set to 0, the user will not be asked to remain silent during calibration and the stt_fx_threshold value will not be changed.
**stt_confidence**

Indicates the minimum difference in confidence required between the top two speech-to-text results for the top result to be accepted. For example, if a grammar contained the terms "faster" and "fastest", the engine may return high confidence values for both words when the user says "faster." If the stt_confidence value is set high and the top two terms return similar confidence values, the engine will reject both results.

Use dissimilar-sounding terms in your grammars to improve speech-to-text results.

**Possible Values**

The default value is 1.

**Example**

```plaintext
Script(Speech_To_Text_Setting_Confidence)
String(strSetting)
String(strDescription)
String(strMessage)
Number(nSettingValue)
Activate(From_Menu)
    strSetting = "stt_confidence"
    nSettingValue = Speech_Get_Setting(strSetting)
    strDescription = Speech_Get_Setting_Value_Desc(strSetting, nSettingValue)
    strMessage = String_Combine(strDescription, "; setting value:")
    strMessage = String_Combine(strMessage, Number_To_String_
    Decimal(nSettingValue))
    Ask_OK(strMessage, strSetting)
    Return
```

**See Also**

stt_expanded, Speech_Get_Setting, Speech_Get_Setting_Value_Desc, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_To_Text, Speech_From_Text, Speech_Setting_Available, Speech_Change_Setting
**stt_expanded**

Returns the confidence value along with the speech-to-text result.

There may be more than one result returned; however, the first result is the one with the highest confidence value. You can use this information to determine the appropriate **stt_threshold** and **stt_confidence** values.

**Possible Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>enabled</td>
</tr>
<tr>
<td>0</td>
<td>disabled</td>
</tr>
</tbody>
</table>

The default value is 0 (disabled).

**Return Value**

If this setting is 1, speech-to-text actions return a string with the top two speech-to-text results and their confidence values.

**Example**

```wscript
Script(Speech_To_Text_Setting_Expanded)
String(strSetting)
String(strDescription)
String(strMessage)
Number(nSettingValue)
Activate(From_Menu)
    strSetting = "stt_expanded"
    nSettingValue = Speech_Get_Setting(strSetting)
    strDescription = Speech_Get_Setting_Value_Desc(strSetting, nSettingValue)
    strMessage = String_Combine(strDescription, "; setting value:")
    strMessage = String_Combine(strMessage, Number_To_String_)
    Ask_OK(strMessage, strSetting)
Return
```

**See Also**

**stt_confidence**, **stt_threshold**, **Speech_Get_Setting**, **Speech_Get_Setting_Value_Desc**, **Speech_Get_Setting_Max**, **Speech_Find_Setting_Value**, **Speech_To_Text**, **Speech_From_Text**, **Speech_Setting_Available**, **Speech_Change_Setting**
**stt_fx_detect_start**

Indicates the action the speech engine should take before attempting to determine what the user is saying. You should leave this setting enabled unless you have specific issues that can only be solved by disabling it.

**Possible Values**

1  The engine will wait until it detects the user is speaking.

0  The engine expects the user to start speaking immediately.

The default value is 1.

**Example**

```plaintext
Script(Speech_To_Text_Setting_Fx_Detect_Start)
String(strSetting)
String(strDescription)
String(strMessage)
Number(nSettingValue)
Activate(From_Menu)
    strSetting = "stt_fx_detect_start"
    nSettingValue = Speech_Get_Setting(strSetting)
    strDescription = Speech_Get_Setting_Value_Desc(strSetting, nSettingValue)
    strMessage = String_Combine(strDescription, "; setting value:")
    strMessage = String_Combine(strMessage, Number_To_String_Decimal(nSettingValue))
    Ask_OK(strMessage, strSetting)
Return
```

**See Also**

`Speech_Get_Setting`, `Speech_Get_Setting_Value_Desc`, `Speech_Get_Setting_Max`, `Speech_Find_Setting_Value`, `Speech_To_Text`, `Speech_From_Text`, `Speech_Setting_Available`, `Speech_Change_Setting`
**stt_fx_microphone**

Tells the speech engine the distance between the user and the microphone. The default value of 0 indicates that the user’s mouth is next to the microphone. A value of 1 should be used if the speaker’s mouth will be located at least several inches away from the microphone.

**Possible Values**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>close</td>
</tr>
<tr>
<td>1</td>
<td>far</td>
</tr>
</tbody>
</table>

The default value is 0.

**Example**

```plaintext
Script(Speech_To_Text_Setting_Fx_Microphone)
String(strSetting)
String(strDescription)
String(strMessage)
Number(nSettingValue)
Activate(From_Menu)
    strSetting = "stt_fx_microphone"
    nSettingValue = Speech_Get_Setting(strSetting)
    strDescription = Speech_Get_Setting_Value_Desc(strSetting, nSettingValue)
    strMessage = String_Combine(strDescription, "; setting value:")
    strMessage = String_Combine(strMessage, Number_To_String_Decimal(nSettingValue))
    Ask_OK(strMessage, strSetting)
Return
```

**See Also**

Speech_Get_Setting, Speech_Get_Setting_Value_Desc, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_To_Text, Speech_From_Text, Speech_Setting_Available, Speech_Change_Setting
**stt_fx_min_duration**

Indicates the minimum duration (in milliseconds) of speech before speech detection is activated. The speech must also have the amount of energy required by the stt_fx_threshold setting.

**Possible Values**

10 ms - 400 ms

The default value is 60 (ms).

**Example**

```plaintext
Script(Speech_To_Text_Setting_Fx_Min_Duration)
String(strSetting)
String(strDescription)
String(strMessage)
Number(nSettingValue)
Activate(From_Menu)
    strSetting = "stt_fx_min_duration"
    nSettingValue = Speech_Get_Setting(strSetting)
    strDescription = Speech_Get_Setting_Value_Desc(strSetting, nSettingValue)
    strMessage = String_Combine(strDescription, "; setting value:")
    strMessage = String_Combine(strMessage, Number_To_String_Decimal(nSettingValue))
    Ask_OK(strMessage, strSetting)
    Return
```

**See Also**

`stt_fx_threshold`, `Speech_Get_Setting`, `Speech_Get_Setting_Value_Desc`, `Speech_Get_Setting_Max`, `Speech_Find_Setting_Value`, `Speech_To_Text`, `Speech_From_Text`, `Speech_Setting_Available`, `Speech_Change_Setting`
**stt_fx_sensitivity**

Indicates the speech detection sensitivity. Wavelink recommends using the calibration tool to adjust this setting, rather than setting it through a script or the screen reformatter. A higher value means speech is more easily detected.

**Possible Values**

Any number from 0 to 100.

The default value is 50. Once the calibration tool has been used, the value it sets will be used as the default for this setting, persisting through a restart of the Client.

**Example**

```plaintext
Script(Speech_To_Text_Setting_Fx_Sensitivity)
String(strSetting)
String(strDescription)
String(strMessage)
Number(nSettingValue)
Activate(From_Menu)
    strSetting = "stt_fx_sensitivity"
    nSettingValue = Speech_Get_Setting(strSetting)
    strDescription = Speech_Get_Setting_Value_Desc(strSetting, nSettingValue)
    strMessage = String_Combine(strDescription, "; setting value:")
    strMessage = String_Combine(strMessage, Number_To_String_Decimal(nSettingValue))
Ask_OK(strMessage, strSetting)
Return
```

**See Also**

`stt_fx_detect_start, Speech_Get_Setting, Speech_Get_Setting_Value_Desc, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_To_Text, Speech_From_Text, Speech_Setting_Available, Speech_Change_Setting`
**stt_fx_silence**

Indicates the milliseconds of silence used to indicate that the user is done speaking.

---

**NOTE:** An acceptable substitute for stt_fx_silence is stt_silence, but stt_fx_silence is preferred.

---

**Possible Values**

0 - 5,000 milliseconds

The default value is 500 (ms).

---

**Example**

```plaintext
Script(Speech_To_Text_Setting_Fx_Silence)
String(strSetting)
String(strDescription)
String(strMessage)
Number(nSettingValue)
Activate(From_Menu)
    strSetting = "stt_fx_silence"
    nSettingValue = Speech_Get_Setting(strSetting)
    strDescription = Speech_Get_Setting_Value_Desc(strSetting, nSettingValue)
    strMessage = String_Combine(strDescription, "; setting value:"
    strMessage = String_Combine(strMessage, Number_To_String(_
      nSettingValue)
    Ask_OK(strMessage, strSetting)
    Return
```

---

**See Also**

stt_timeout, stt_idle_timeout, Speech_Get_Setting, Speech_Get_Setting_Value_Desc, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_To_Text, Speech_From_Text, Speech_Setting_Available, Speech_Change_Setting
**stt_fx_threshold**

Indicates the amount of energy the microphone input must have before the speech detection (stt_fx_detect_start) is activated. Wavelink recommends using the calibration tool to adjust this setting, rather than setting it through a script or the screen reformatter. Changing this value using a script or the Screen Reformatter will not change the default value.

**Possible Values**

0 (-72dB) to 9000 (18dB)

The default value is 2200 (-50dB). Each increase of 100 is equal to 1dB. Once the calibration tool has been used, the value it sets will be used as the default for this setting, persisting through a restart of the Client.

**Example**

```plaintext
Script(Speech_To_Text_Setting_Fx_Threshold)
String(strSetting)
String(strDescription)
String(strMessage)
Number(nSettingValue)
Activate(From_Menu)
    strSetting = "stt_fx_threshold"
    nSettingValue = Speech_Get_Setting(strSetting)
    strDescription = Speech_Get_Setting_Value_Desc(strSetting,
        nSettingValue)
    strMessage = String.Combine(strDescription, "; setting value:")
    strMessage = String.Combine(strMessage, Number.To_String_Decimal(nSettingValue))
    Ask_OK(strMessage, strSetting)
Return
```

**See Also**

stt_fx_detect_start, Speech_Get_Setting, Speech_Get_Setting_Value_Desc, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_To_Text, Speech_From_Text, Speech_Setting_Available, Speech_Change_Setting
**stt_idle_timeout**

Indicates the time in milliseconds for the engine to continue collecting results following the last result or stt_timeout. If any changes (settings, grammar, etc.) are made during the stt_idle_timeout period, the results generated during the period will be discarded.

**Possible Values**

The default value is 2000 ms (2 seconds).

**Example**

```plaintext
Script(Speech_To_Text_Setting_Idle_Timeout)
String(strSetting)
String(strDescription)
String(strMessage)
Number(nSettingValue)
Activate(From_Menu)
    strSetting = "stt_idle_timeout"
    nSettingValue = Speech_Get_Setting(strSetting)
    strDescription = Speech_Get_Setting_Value_Desc(strSetting, nSettingValue)
    strMessage = String_Combine(strDescription, "; setting value:" )
    strMessage = String_Combine(strMessage, Number_To_String_Decimal(nSettingValue))
Ask_OK(strMessage, strSetting)
Return
```

**See Also**

stt_timeout, Speech_Get_Setting, Speech_Get_Setting_Value_Desc, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_To_Text, Speech_From_Text, Speech_Setting_Available, Speech_Change_Setting
**stt_language_long**

Displays the full name of the language currently being used. For example, English USA instead of ENU.

For possible language values, see **tts_language_short** on page 318.

---

**Example**

```plaintext
Script(Speech_To_Text_Setting_Language_Long)
String(strSetting)
String(strDescription)
String(strMessage)
Number(nSettingValue)
Activate(From_Menu)
    strSetting = "stt_language_long"
    nSettingValue = Speech_Get_Setting(strSetting)
    strDescription = Speech_Get_Setting_Value_Desc(strSetting, nSettingValue)
    strMessage = String_Combine(strDescription, "; setting value:")
    strMessage = String_Combine(strMessage, Number_To_String_Decimal(nSettingValue))
    Ask_OK(strMessage, strSetting)
Return
```

---

**See Also**

stt_language_short

Displays the three-letter abbreviation of the language currently being used.

For possible language values, see tts_language_short on page 318.

**NOTE:** An acceptable substitute for stt_language_short is stt_language, but stt_language_short is preferred.

**Example**

```plaintext
Script(Speech_To_Text_Setting.Language.Short)
String(strSetting)
String(strDescription)
String(strMessage)
Number(nSettingValue)
Activate(From_Menu)
    strSetting = "stt_language_short"
    nSettingValue = Speech_Get_Setting(strSetting)
    strDescription = Speech_Get_Setting_Value_Desc(strSetting, nSettingValue)
    strMessage = String_Combine(strDescription, "; setting value:"))
    strMessage = String_Combine(strMessage, Number_To_String_Decimal(nSettingValue))
    Ask_OK(strMessage, strSetting)
Return
```

**See Also**

stt_language_long, Speech_Get_Setting, Speech_Get_Setting_Value_Desc, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_To_Text, Speech_From_Text, Speech_Setting_Available, Speech_Change_Setting
**stt_logging**

If enabled, the speech-to-text engine will create a file with information that can be useful for diagnostics. Logging information is written to the \Temp\STT_Log.txt file.

**Possible Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>disabled</td>
</tr>
<tr>
<td>1</td>
<td>critical</td>
</tr>
<tr>
<td>2</td>
<td>informational</td>
</tr>
<tr>
<td>3</td>
<td>verbose</td>
</tr>
</tbody>
</table>

The default value is 0 (disabled).

**Example**

```plaintext
Script( Speech_Logging )
Activate( From_Menu )
    If( Ask,Yes_No( "Do you want to use Speech Logging?", "Logging", FALSE ) )
        Speech_Change_Setting( "stt_logging", 1 )
        Message( "Logging is enabled.", 5 )
    Else
        Speech_Change_Setting( "stt_logging", 0 )
        Message( "Logging is disabled.", 5 )
    End_If
Return
```
**stt logging audio**

This setting allows the user to create a .wav file of speech-to-text attempts. The audio capture files will be written to the \Temp folder. Because of the amount of additional processing and memory required, this setting should only be used on fast devices for diagnostic purposes.

If the device runs out of memory, logging will be disabled.

**Possible Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No audio logging.</td>
</tr>
<tr>
<td>1</td>
<td>Log audio data until the microphone is closed. When the speech-to-text attempt is complete, this setting will reset to 0.</td>
</tr>
<tr>
<td>2</td>
<td>Log audio data, creating a new file each time the microphone is closed.</td>
</tr>
<tr>
<td>3</td>
<td>Log audio data until the microphone is closed, a speech-to-text result is returned, or an abnormal condition occurs. When the audio capture is complete, the setting will reset to 0.</td>
</tr>
<tr>
<td>4</td>
<td>Log audio data, creating a new file each time the microphone is closed, a speech-to-text result is returned, or an abnormal condition occurs.</td>
</tr>
</tbody>
</table>

The default value is 0.

**See Also**

stt logging, stt logging engine
**stt_logging_engine**

Sets the engine to log speech-to-text attempts as .wav files. This file contains information that can be useful for diagnostics. The default value is 0 (disabled).

**Possible Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>disabled</td>
</tr>
<tr>
<td>1</td>
<td>enabled</td>
</tr>
</tbody>
</table>

**See Also**

* stt_logging, stt_logging_audio*
**stt_pool_size**

Sets the number of terms the engine will examine closely for the best match. The engine does a rough match for the grammar file, then compares the best matches with the speech to determine which result to return. This setting is for large grammar files that may have many possible results. Since a higher value requires more resources, increasing this parameter should only be done on machines that have more CPU and memory resources such as a PC or new generation mobile device.

**Possible Values**

5-1000

The default value is 5 for mobile devices, and 10 for desktop or laptop computers.
**stt_preserve**

Causes the engine state to be saved. If the stt_preserve setting has been used, the saved session state will no longer be overwritten during that session.

**Possible Value**

0  
Causes the speech engine to save the current engine state for use later. Use this setting if the speech recognition is working very well and you don't want the engine to continue adapting.

**Example**

```plaintext
Script( Speech_To_Text_Preserve )
Activate( From_Menu )
    If( Ask.Yes_No( "Is the speech recognition performing great?", "Preserve", FALSE ) )
        Speech_Change_Setting( "stt_preserve", 0 )
        Message( "Saving the current state.", 5 )
    Else
        Message( "No action taken.", 5 )
    End_If
Return
```
**stt_priority**

Determines how aggressively the microphone input is collected and speech analysis is performed.

You can increase the priority if the results are taking a long time to process, and decrease the priority if you experience issues such as the device locking up or network connections being dropped during speech-to-text processing.

### Possible Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>low</td>
</tr>
<tr>
<td>1</td>
<td>medium</td>
</tr>
<tr>
<td>2</td>
<td>high</td>
</tr>
<tr>
<td>3</td>
<td>critical</td>
</tr>
</tbody>
</table>

The default value is 1 (medium).

### Example

```plaintext
Script(Speech_To_Text_Setting_Priority)
String(strSetting)
String(strDescription)
String(strMessage)
Number(nSettingValue)
Activate(From_Menu)
    strSetting = "stt_priority"
    nSettingValue = Speech_Get_Setting(strSetting)
    strDescription = Speech_Get_Setting_Value_Desc(strSetting, nSettingValue)
    strMessage = String_Combine(strDescription, "; setting value:")
    strMessage = String_Combine(strMessage, Number_To_String_
Decimal(nSettingValue))
    Ask_OK(strMessage, strSetting)
    Return
```

### See Also

`Speech_Get_Setting`, `Speech_Get_Setting_Value_Desc`, `Speech_Get_Setting_Max`, `Speech_Find_Setting_Value`, `Speech_To_Text`, `Speech_From_Text`, `Speech_Setting_Available`, `Speech_Change_Setting`
stt_processing

Indicates the action the speech engine should take when returning a grammar result.

If the setting is 1, the speech engine will return the semantic result (if available) instead of the actual phrase spoken by the user.

This setting is useful for grammars that incorporate bracket ({} ) directives.

This setting will enable and disable both Actions (the !action directive) and semantic ("result" and "@") processing.

Possible Values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>on</td>
</tr>
<tr>
<td>0</td>
<td>off</td>
</tr>
</tbody>
</table>

The default value is 1 (on/enabled).

Example

```script
Script(Speech_To_Text_Setting_Processing)
String(strSetting)
String(strDescription)
String(strMessage)
Number(nSettingValue)
Activate(From_Menu)
    strSetting = "stt_processing"
    nSettingValue = Speech_Get_Setting(strSetting)
    strDescription = Speech_Get_Setting_Value_Desc(strSetting, nSettingValue)
    strMessage = String_Combine(strDescription, "; setting value:" )
    strMessage = String_Combine(strMessage, Number_To_String_Decimal(nSettingValue))
    Ask_OK(strMessage, strSetting)
Return
```

See Also

Speech_Get_Setting, Speech_Get_Setting_Value_Desc, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_To_Text, Speech_From_Text, Speech_Setting_Available, Speech_Change_Setting
**stt_reset**

An action that resets the speech engine. The engine may need to be reset if it has adapted poorly (e.g., if the engine is listening for prolonged periods when the user is not speaking, it will try to adapt to the ambient noise).

**Possible Values**

- **0** (soft reset) Temporarily increase the engine adaptation speed.
- **1** (hard reset) Erase any saved state information and temporarily increase the adaptation speed.
- **2** Restore the last saved state without increasing the adaptation speed.

**Example**

```plaintext
Script( Speech_To_Text_Reset )
Activate( From_Menu )
    If( Ask_Yes_No( "Is the speech-to-text behaving very poorly?", "Speech-to-Text", TRUE ) )
        Message( "Clearing the speech engine state.", 5 )
        Speech_Change_Setting( "stt_reset", 1 )
    Else
        If( Ask_Yes_No( "Has the user changed?", "Speech User", FALSE ) )
            Message( "Increasing the adaptation rate.", 5 )
            Speech_Change_Setting( "stt_reset", 0 )
        Else
            Message( "Reverting to the last saved state.", 5 )
            Speech_Change_Setting( "stt_reset", 2 )
        End_If
    End_If
End_If
Return
```
**stt_reset_session_delay**

Indicates the amount of time in seconds for the speech engine to wait for a valid response before reverting back to the last saved engine state. This setting prevents the performance from degrading if the user does not speak for a long period of time.

**Possible Values**

The default value is 30 (seconds).

**Example**

```plaintext
Script(Speech_To_Text_Setting_Reset_Session_Delay)
String(strSetting)
String(strDescription)
String(strMessage)
Number(nSettingValue)
Activate(From_Menu)
    strSetting = "stt_reset_session_delay"
    nSettingValue = Speech_Get_Setting(strSetting)
    strDescription = Speech_Get_Setting_Value_Desc(strSetting, nSettingValue)
    strMessage = String_Combine(strDescription, "; setting value:"
                  strMessage = String_Combine(strMessage, Number_To_String_
    Decimal(nSettingValue))
    Ask_OK(strMessage, strSetting)
Return
```

**See Also**

stt_save_session_delay, Speech_Get_Setting, Speech_Get_Setting_Value_Desc, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_To_Text, Speech_From_Text, Speech_Setting_Available, Speech_Change_Setting
stt_result_sound

Determines when a result sound is played for speech-to-text.

Possible Values

2 Two sounds are used: one for successful recognitions, one for unsuccessful recognitions

1 A sound is played for each successful recognition

0 No result sounds are used

Example

Script( Speech_To_Text_Result_Sound )
Activate( From_Menu )
   If( Ask,Yes_No( "Do you want to hear a beep on a good recognition?", "Beep", FALSE ) )
      If( Ask,Yes_No( "Do you want to hear a beep on a bad recognition?", "Beep", FALSE ) )
         Speech_Change_Setting( "stt_result_sound", 2 )
         Message( "Result sounds enabled.", 5 )
      Else
         Speech_Change_Setting( "stt_result_sound", 1 )
         Message("Result sounds only played for good recognitions.", 5)
      End_If
   Else
      Speech_Change_Setting( "stt_result_sound", 0 )
      Message( "Result sounds disabled.", 5 )
   End_If
Return
**stt_save_increase**

How much the session save threshold will increase each minute since the last cold restart (stt_reset of 1). The default value is 100, meaning that the save threshold will increase by 100 for every minute the engine is used. This parameter is used to increase the threshold for saving a new engine state as time progresses, since the engine state can start to degrade after speech-to-text recognition has been used for long periods.

For example, if using the following settings:
```plaintext
stt_threshold=4500, stt_save_threshold=1000, stt_save_increase=100
```
the save threshold would be 5500 during the first minute following a hard reset (4500+1000+(0*100)). A speech-to-text result with a confidence level of 5500 would match the save threshold and cause the engine state to save. After 8 minutes, a speech-to-text result with a confidence of 6300 (4500+1000+(8*100)) would be required to save the engine state.

When the engine state is restored (using stt_reset_session_delay), it will revert to the last time the engine state was saved. If stt_reset is set to 0 (a soft reset), the threshold will be reduced to account for the accelerated learning that will follow.

If you still find that the recognition rate is decreasing over long periods of time (such as an hour or more), you can make the stt_save_threshold larger. If you are in environments where the background sounds can change, you may want to make stt_save_threshold smaller. Set stt_save_threshold to 0 to disable this feature.

**Example**

```plaintext
Script( Speech_To_Text_Save_Increase )
String( strSetting )
String( strDescription )
String( strMessage )
Number( nSettingValue )
Activate( From_Menu )
    strSetting = "stt_save_increase"
    nSettingValue = Speech_Get_Setting( strSetting )
    strDescription = Speech_Get_Setting_Value_Desc( strSetting, nSettingValue )
    strMessage = String_Combine( strDescription, "; setting value:" )
    strMessage = String_Combine( strMessage, Number_To_String.Decimal( nSettingValue ) )
    Ask.OK( strMessage, strSetting )
Return
```
**stt_save_session_delay**

Indicates the number of seconds for the speech engine to wait before saving the engine state again. For example, if the engine state was saved 5 seconds ago, and this value is set to 10 seconds, it will not save the engine state again for at least 5 more seconds. The engine will save its state regularly, allowing it to adapt to the speaker and revert back to a saved state if necessary.

**Possible Values**

The default value is 10 (seconds).

**Example**

```plaintext
Script(Speech_To_Text_Setting_Save_Session_Delay)
String(strSetting)
String(strDescription)
String(strMessage)
Number(nSettingValue)
Activate(From_Menu)
    strSetting = "stt_save_session_delay"
    nSettingValue = Speech_Get_Setting(strSetting)
    strDescription = Speech_Get_Setting_Value_Desc(strSetting, nSettingValue)
    strMessage = String_Combine(strDescription, "; setting value:")
    strMessage = String_Combine(strMessage, Number_To_String_
    Ask_OK(strMessage, strSetting)
Return
```

**See Also**

`stt_reset_session_delay, Speech_Get_Setting, Speech_Get_Setting_Value_Desc, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_To_Text, Speech_From_Text, Speech_Setting_Available, Speech_Change_Setting`
**stt_save_threshold**

Directs the speech engine to save the engine state if the result confidence is greater than the values of stt_threshold and stt_save_threshold combined.

When the engine is getting high confidence results, it will save the engine state. If engine adaptation causes results to degrade, you can revert to a saved engine state by using stt_reset or stt_reset_session_delay.

**Possible Values**

The default value is 2000.

**Example**

```plaintext
Script(Speech_To_Text_Setting_Save_Threshold)
String(strSetting)
String(strDescription)
String(strMessage)
Number(nSettingValue)
Activate(From_Menu)
   strSetting = "stt_save_threshold"
   nSettingValue = Speech_Get_Setting(strSetting)
   strDescription = Speech_Get_Setting_Value_Desc(strSetting, nSettingValue)
   strMessage = String_Combine(strDescription, "; setting value:"
   strMessage = String_Combine(strMessage, Number_To_String_Decimal(nSettingValue))
   Ask_OK(strMessage, strSetting)
   Return
```

**See Also**

stt_threshold, Speech_Get_Setting, Speech_Get_Setting_Value_Desc, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_To_Text, Speech_From_Text, Speech_Setting_Available, Speech_Change_Setting
stt_server_timeout

When uploading or downloading user training data, the stt_server_timeout value is how long (in seconds) the Client will wait for a response from the Avalanche server.

Possible Values

0-60. If the value is set to 0, the Client will not attempt to contact the Avalanche server. The default value is 15.

Example

Script( Speech_Server_Timeout )
Number( nTimeout )
Activate( From_Menu )
    nTimeout = Speech_Get_Setting( "stt_server_timeout" )
    If( Number_LessThan( nTimeout, 0 ) )
        Message( "Server timeout is not supported.", 5 )
        Return
    End_If

    nTimeout = Ask_Number( "How long do you want the server timeout to be (in seconds)?", "Server Timeout", 1, 60, nTimeout )
    Speech_Change_Setting( "stt_server_timeout", nTimeout )
    Message( "Changed the server timeout.", 5 )
    Return
**stt_size**

Displays the size of the speech-to-text engine being used.

**Possible Values**

- Full
- Compact

**Example**

```plaintext
Script(Speech_To_Text_Setting_Size)

String(strSetting)
String(strDescription)
String(strMessage)
Number(nSettingValue)
Activate(From_Menu)
    strSetting = "stt_size"
    nSettingValue = Speech_Get_Setting(strSetting)
    strDescription = Speech_Get_Setting_Value_Desc(strSetting, nSettingValue)
    strMessage = String_Combine(strDescription, "; setting value:" )
    strMessage = String_Combine(strMessage, Number_To_String_Decimal(nSettingValue))
    Ask_OK(strMessage, strSetting)
Return
```

**See Also**

- `Speech_Get_Setting`
- `Speech_Get_Setting_Value_Desc`
- `Speech_Get_Setting_Max`
- `Speech_Find_Setting_Value`
- `Speech_To_Text`
- `Speech_From_Text`
- `Speech_Setting_Available`
- `Speech_Change_Setting`
**stt_special_sounds**

Indicates how the speech engine should interpret special sounds.

If the setting is 1, the speech engine will examine sounds to determine if they are more likely to correspond to a special sound (empty pauses, coughing, etc.) than a valid grammar result.

If your grammar consists mostly of multi-syllable words or phrases, enabling this setting will result in fewer low-confidence results. However, enabling this setting may result in one- or two-syllable words (such as "yes," "two," etc.) being rejected.

**Possible Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>on</td>
</tr>
<tr>
<td>0</td>
<td>off</td>
</tr>
</tbody>
</table>

The default value is 0 (off).

**Example**

```plaintext
Script(Speech_To_Text_Setting_Special_Sounds)
String(strSetting)
String(strDescription)
String(strMessage)
Number(nSettingValue)
Activate(From_Menu)
    strSetting = "stt_special_sounds"
    nSettingValue = Speech_Get_Setting(strSetting)
    strDescription = Speech_Get_Setting_Value_Desc(strSetting, nSettingValue)
    strMessage = String_Combine(strDescription, "; setting value:"")
    strMessage = String_Combine(strMessage, Number_To_String_Decimal(nSettingValue))
Decimal(nSettingValue)
    Ask_OK(strMessage, strSetting)
Return
```

**See Also**

`Speech_Get_Setting`, `Speech_Get_Setting_Value_Desc`, `Speech_Get_Setting_Max`, `Speech_Find_Setting_Value`, `Speech_To_Text`, `Speech_From_Text`, `Speech_Setting_Available`, `Speech_Change_Setting`
**stt_threshold**

Indicates the minimum amount of confidence for the most-likely result that will be accepted. If the confidence is less than the set value, the result will be discarded and the speech-to-text action will report that it failed.

You may want to use different values for different grammars.

**Possible Values**

0-10000.

The default value is 5000.

**Example**

```plaintext
Script(Speech_To_Text_Setting_Threshold)
String(strSetting)
String(strDescription)
String(strMessage)
Number(nSettingValue)
Activate(From_Menu)
    strSetting = "stt_threshold"
    nSettingValue = Speech_Get_Setting(strSetting)
    strDescription = Speech_Get_Setting_Value_Desc(strSetting, nSettingValue)
    strMessage = String_Combine(strDescription, "; setting value:"")
    strMessage = String_Combine(strMessage, Number_To_String_Decimal(nSettingValue))
Ask_OK(strMessage, strSetting)
Return
```

**See Also**

* stt_expanded, Speech_Get_Setting, Speech_Get_Setting_Value_Desc, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_To_Text, Speech_From_Text, Speech_Setting_Available, Speech_Change_Setting
### stt_timeout

Indicates the time in milliseconds for the system to wait before responding that no speech was detected.

#### Possible Values

0 - 300,000 milliseconds

The default value is 10000 ms (10 seconds).

#### Example

```plaintext
Script(Speech_To_Text_Setting_Timeout)
String(strSetting)
String(strDescription)
String(strMessage)
Number(nSettingValue)
Activate(From_Menu)
    strSetting = "stt_timeout"
    nSettingValue = Speech_Get_Setting(strSetting)
    strDescription = Speech_Get_Setting_Value_Desc(strSetting, nSettingValue)
    strMessage = String_Combine(strDescription, "; setting value:"
    strMessage = String_Combine(strMessage, Number_To_String_Decimal(nSettingValue))
Ask_OK(strMessage, strSetting)
Return
```

#### See Also

`stt_idle_timeout`, `stt_fx_silence`, `Speech_Get_Setting`, `Speech_Get_Setting_Value_Desc`, `Speech_Get_Setting_Max`, `Speech_Find_Setting_Value`, `Speech_To_Text`, `Speech_From_Text`, `Speech_Setting_Available`, `Speech_Change_Setting`, `Speech_To_Text_No_Wait`
**stt_use_jumpback**

Sets a buffer to check if the engine is processing speech. If this setting is changed to 0 (disabled), then the engine will use less memory, but will use more CPU cycles in situations where the engine gets a false positive when the user is not speaking.

### Possible Values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>disabled</td>
</tr>
<tr>
<td>1</td>
<td>enabled</td>
</tr>
</tbody>
</table>

The default value for this setting is 1 (enabled).

### Example

```
stt_use_jumpback(0)
```
**stt_use_word_ids**

This feature enables support for word IDs (the !id directive) in grammar files. When a term has a word ID assigned in the grammar file, the engine will return a number (converted to a string) rather than the speech result, which can be useful for localization.

For example, if the grammar file specified:
`abort !id(4)`
then when the user said "abort," the engine would return "4".

**Possible Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Ignores !id directives in grammar files.</td>
</tr>
<tr>
<td>1</td>
<td>Returns the word ID (a number) specified in the grammar for a term.</td>
</tr>
</tbody>
</table>

The default value is 1.

**Example**

```plaintext
Script( word_ids_example )
Number( nUseWordIds )
Activate( From_Menu )
    nUseWordIds = Speech_Get_Setting( "stt_use_word_ids" )
    If( Number_Equal( nUseWordIds, 1 ) )
        Message( "!id values in grammars will be used.", 5 )
    Else
        Message( "!id values in grammars will be ignored.", 5 )
    End_If
Return
```
**stt_volume**

Indicates the current volume of the microphone input. Wavelink recommends using the calibration tool to adjust this setting, rather than setting it through a script or the screen reformatter.

This setting is not supported by all mobile devices.

**Possible Values**

Any number from 0 (lowest) to 100 (highest).

**Example**

```plaintext
Script(Speech_To_Text_Setting_Volume)
String(strSetting)
String(strDescription)
String(strMessage)
Number(nSettingValue)
Activate(From_Menu)
    strSetting = "stt_volume"
    nSettingValue = Speech_Get_Setting(strSetting)
    strDescription = Speech_Get_Setting_Value_Desc(strSetting, nSettingValue)
    strMessage = String_Combine(strDescription, "; setting value:"
    strMessage = String_Combine(strMessage, Number_To_String_Decimal(nSettingValue))
    Ask_OK(strMessage, strSetting)
Return
```

**See Also**

*stt_calibrate, stt_fx_microphone, Speech_Get_Setting, Speech_Get_Setting_Value_Desc, Speech_Get_Setting_Max, Speech_Find_Setting_Value, Speech_To_Text, Speech_From_Text, Speech_Setting_Available, Speech_Change_Setting*
Symbologies and Values

The following is a list of symbologies and their values:

<table>
<thead>
<tr>
<th>Symbology</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPCE0</td>
<td>48</td>
</tr>
<tr>
<td>UPCE1</td>
<td>49</td>
</tr>
<tr>
<td>UPCA</td>
<td>50</td>
</tr>
<tr>
<td>MSI</td>
<td>51</td>
</tr>
<tr>
<td>EAN8</td>
<td>52</td>
</tr>
<tr>
<td>EAN13</td>
<td>53</td>
</tr>
<tr>
<td>CODABAR</td>
<td>54</td>
</tr>
<tr>
<td>CODE 39</td>
<td>55</td>
</tr>
<tr>
<td>D 2 OF 5</td>
<td>56</td>
</tr>
<tr>
<td>I 2 OF 5</td>
<td>57</td>
</tr>
<tr>
<td>CODE 11</td>
<td>58</td>
</tr>
<tr>
<td>CODE 93</td>
<td>59</td>
</tr>
<tr>
<td>CODE 128</td>
<td>60</td>
</tr>
<tr>
<td>D 2 OF 5 IATA</td>
<td>62</td>
</tr>
<tr>
<td>EAN/UCC 128</td>
<td>63</td>
</tr>
<tr>
<td>PDF417</td>
<td>64</td>
</tr>
<tr>
<td>TRIOPTIC 39</td>
<td>66</td>
</tr>
<tr>
<td>COUPON CODE</td>
<td>67</td>
</tr>
<tr>
<td>BOOKLAND</td>
<td>68</td>
</tr>
<tr>
<td>MICROPDF</td>
<td>69</td>
</tr>
<tr>
<td>CODE 32</td>
<td>70</td>
</tr>
<tr>
<td>MACRO PDF</td>
<td>71</td>
</tr>
<tr>
<td>Symbology</td>
<td>Value</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>MAXOCODE</td>
<td>72</td>
</tr>
<tr>
<td>DATAMATRIX</td>
<td>73</td>
</tr>
<tr>
<td>QR CODE</td>
<td>74</td>
</tr>
<tr>
<td>MACRO_MICROPDF</td>
<td>75</td>
</tr>
<tr>
<td>RSS 14</td>
<td>76</td>
</tr>
<tr>
<td>RSS LIMITED</td>
<td>77</td>
</tr>
<tr>
<td>RSS EXPANDED</td>
<td>78</td>
</tr>
<tr>
<td>SIGNATURE</td>
<td>82</td>
</tr>
<tr>
<td>WEBCODE</td>
<td>84</td>
</tr>
<tr>
<td>CUECODE</td>
<td>85</td>
</tr>
<tr>
<td>COMPOSITE</td>
<td>86</td>
</tr>
<tr>
<td>TLC 39</td>
<td>88</td>
</tr>
<tr>
<td>POSTNET</td>
<td>97</td>
</tr>
<tr>
<td>PLANET</td>
<td>98</td>
</tr>
<tr>
<td>BRITISH POSTAL</td>
<td>99</td>
</tr>
<tr>
<td>JAPAN POSTAL</td>
<td>100</td>
</tr>
<tr>
<td>AUSTRALIA POSTAL</td>
<td>101</td>
</tr>
<tr>
<td>DUTCH POSTAL</td>
<td>102</td>
</tr>
<tr>
<td>CANADA POSTAL</td>
<td>103</td>
</tr>
<tr>
<td>AZTEC</td>
<td>160</td>
</tr>
<tr>
<td>AZTEC MESA</td>
<td>161</td>
</tr>
<tr>
<td>CODE 49</td>
<td>162</td>
</tr>
<tr>
<td>OCR</td>
<td>163</td>
</tr>
<tr>
<td>CODABLOCK</td>
<td>164</td>
</tr>
<tr>
<td>MATRIX 2 OF 5</td>
<td>165</td>
</tr>
<tr>
<td>Symbology</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------</td>
</tr>
<tr>
<td>PLESSEY</td>
<td>166</td>
</tr>
<tr>
<td>CHINA POSTAL</td>
<td>167</td>
</tr>
<tr>
<td>KOREA POSTAL</td>
<td>168</td>
</tr>
<tr>
<td>TELEPEN</td>
<td>169</td>
</tr>
<tr>
<td>CODE 16K</td>
<td>170</td>
</tr>
<tr>
<td>POSICODE</td>
<td>171</td>
</tr>
<tr>
<td>UPC</td>
<td>241</td>
</tr>
<tr>
<td>MSR</td>
<td>245</td>
</tr>
<tr>
<td>RFID</td>
<td>246</td>
</tr>
</tbody>
</table>
Examples

This appendix provides example scripts. Use these script examples to customize your own scripts according to preference.

- Beep Sample Script
- Escape Sequence Sample Script
- Request Information Sample Script
- Display Screen Button Sample Script
- Get_Number Sample Script
- Get_Number_Test Sample Script
- Play_Screen Sample Script
- Speech_Button_Demo Sample Script

Beep Sample Script

This is an example of a script that tells the device to beep if the word ALARM appears on the top five rows of the screen.

Example Code

```plaintext
If_Not(Search_Screen("ALARM",1,5,FALSE))
  Return
End_If
Beep(1000,200,5)
Delay(200)
Beep(1500,500,9)
Return
```

Notes

This example should be set to activate each time the screen changes. Make sure that the ALARM text disappears quickly after being shown. Otherwise, the alarm will go off each time the screen updates (because the user pressed a key, each character from a bar code scanned was shown on the screen, etc.).

Here is an alternate implementation that waits for the ALARM text to disappear. The limitation with this version is that no other scripts will be able to run until the ALARM text is removed from the screen.

```plaintext
If_Not(Search_Screen("ALARM",1,5,FALSE))
  Return
```
Examples

End_If
Beep(1000,200,5)
Delay(200)
Beep(1500,500,9)
While(Search_Screen("ALARM",1,5,FALSE))
Wait_For_Screen_Update
End_While

Escape Sequence Sample Script

This is an example of using an escape sequence to turn off the Codabar symbology.

Example Code

Result=Escape_Sequence("%8D")
Return

Notes

You need to create a string variable named Result for this example, since the Escape_Sequence command returns a string value.

Refer to Creating Variables on page 25 for more information on creating variables.

Request Information Sample Script

This example requests the mobile device user to input some information and displays the result.

Example Code

Result=Ask_String("Enter a string:","Length Calculator",0,200,"")
Ask_OK(String_Combine("The string",String_Combine( Result, String_Combine("is",String_Combine( Number_To_String_Decimal(String_Length(Result)),"characters long." ))),"String Length")
Return

Notes

This example also requires a string variable named Result. The ASK_OK instruction uses actions inside of actions to get several layers deep. You could also use variables to break that instruction into several short instructions.
Display Screen Button Sample Script

This example displays a log out button that appears in the in the bottom-left corner of the screen (row two, column five) when the text "logged out" appears on the screen. Pressing the button allows you to exit the Client.

Example Code

```plaintext
If_Not(String_Equal( Get_Screen_Text_Columns(2,5,10), "logged out", 0, FALSE ))
  Return
End_If
Button_Create_View( "Exit", 1000,1,0,ButtonPressed)
While_Not( ButtonPressed)
  If_Not(String_Equal(Get_Screen_Text_Columns(2,5,10), "logged out",0,FALSE))
    Return
  End_If
  Wait_For_Screen_Update
End_While
Exit_Application(0)
```

Notes

This example uses a boolean variable ButtonPressed to know if the screen button is pressed. The button is destroyed when the script exits so you do not need to delete the script.

The While_Not loop uses the Wait_For_Screen_Update action to detect if the logged out text is no longer there so that the scripts do not continually loop.

Get_Number Sample Script

The following example script is called by the Get_Number_Test script. It retrieves the appropriate number for the Get_Number_Test script to display.

```plaintext
Comment:This script is designed to be called by other scripts.
Comment:The result of the Speech-to-Text will be in the nResult variable.
Comment:The number.bnf file must be available as a grammar file.
Speech_To_Text(sResult,"number")
nResult=0
While_Not(String_Empty(sResult))
nNextSpace=String_Find_First(sResult,"",FALSE)
```
nResult=Number_Plus(nResult,String_To_Number.Decimal(sResult))
If_Number_Less_Than(nNextSpace,0))
    Break
End_If
nNextSpace=Number_Plus(nNextSpace,1)
sResult=String_Right(sResult,Number_Minus
(String_Length(sResult),nNextSpace))
End_While
Return

Get_Number_Test Sample Script

The following example script converts a spoken number into text that displays on the mobile device. This script must be used in conjunction with the Get_Number sample script.

Speech_From_Text("Say a number",FALSE)
Call:Get_Number
    nResult <-> nResult
Ask_OK(Number_To_String.Decimal(nResult),
    "Number Returned")
Return

Play_Screen Sample Script

The following example script converts the current TE Client screen into speech that the user can hear.

nNumRows=Get_Screen_Rows
nCurrentRow=1
While(Number_Less_Than_Or_Equal(nCurrentRow,nNumRows))
    Speech_From_Text(Get_Screen_Text
    (nCurrentRow,1),FALSE)
    nCurrentRow=Number_Plus(nCurrentRow,1)
End_While
Return

Speech_Button_Demo Sample Script

The following example script creates the following buttons on the screen: Digits, State, Play Screen, Done. When selected, the buttons allow the user to verbally input data.

While_Not(bExit)
    WaitForScreen_Update
If(bPlayScreen)
    bPlayScreen=FALSE
    Button_Remove_All
    bButtonsVisible=FALSE
    Delay(1)
    nNumRows=Get_Screen_Rows
    nCurrentRow=1
    While(Number_Less_Than_Or_Equal(nCurrentRow,nNumRows))
        Speech_From_Text(Get_Screen_Text (nCurrentRow,1),FALSE)
        nCurrentRow=Number_Plus(nCurrentRow,1)
    End_While
End_If
If(bGetDigits)
    bGetDigits=FALSE
    Button_Remove_All
    bButtonsVisible=FALSE
    Message("Say 1 or more digits...",0)
    szResult=""
    Speech_To_Text(szResult,"connected_digits")
    Message_Clear
    szResult=String_Strip_Characters(szResult,"",FALSE)
    Keypress_String(szResult)
End_If
If(bGetState)
    bGetState=FALSE
    Button_Remove_All
    bButtonsVisible=FALSE
    Message("Say a USA state...",0)
    szResult=""
    Speech_To_Text(szResult,"usa_states")
    Message_Clear
    Keypress_String(szResult)
End_If
End_While
Button_Remove_All
Return
**Script Build Errors**

This appendix describes the error messages that appear in the *Script Editor Error Help* dialog box. For more information about accessing this dialog box, see Building Scripts with the Text Editor on page 34.

<table>
<thead>
<tr>
<th>Error Number</th>
<th>Error Message</th>
<th>Description</th>
<th>Fixing the Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>WL1001</td>
<td>You should use two backslash characters (&quot;\&quot;) to represent a single backslash character.</td>
<td>Use the string &quot;\&quot; to represent the backslash character.</td>
<td>Add one more backslash in the string, next to the single backslash that is already there.</td>
</tr>
<tr>
<td>WL1002</td>
<td>Not enough parameter values for the action; there must be at least two.</td>
<td>The actions Boolean_And() and Boolean_Or() should have two or more parameters. Each of the actions can have up to five parameters.</td>
<td>Pass 2, 3, 4, or 5 parameters to Boolean_And() and to Boolean_or().</td>
</tr>
<tr>
<td>WL1003</td>
<td>Call to non-existing script.</td>
<td>This error occurs if the name doesn’t exactly match the name of a script or if the script does not exist.</td>
<td>Either call a script that exists or write the non-existing script. Make sure that the name of the script matches exactly, including the case of the letters.</td>
</tr>
<tr>
<td>WL1004</td>
<td>Division by 0 is not allowed.</td>
<td>The divisor passed to Number_Divide() is zero, and dividing by zero is an undefined mathematical operation.</td>
<td>Make sure to never use 0 as the divisor in Number_Divide().</td>
</tr>
<tr>
<td>WL1005</td>
<td>No variable assigned to the value for this action.</td>
<td>The action returns a value, but the script is not using that value.</td>
<td>Create a variable and assign the action’s value to the variable.</td>
</tr>
<tr>
<td>WL1006</td>
<td>Call to script references non-existing (or incorrect type) variable.</td>
<td>The variable that is being exchanged with the Call action cannot be found in the called script, or the variable type doesn’t match the type in the called script.</td>
<td>Reference a variable that exists or a variable that is the correct type.</td>
</tr>
<tr>
<td>WL1007</td>
<td>The host profile does not exist.</td>
<td>The script activates on a host profile, but that host profile does not exist because the</td>
<td>Either use a host profile that exists or create the host profile. Make sure that the</td>
</tr>
<tr>
<td>Error Number</td>
<td>Error Message</td>
<td>Description</td>
<td>Fixing the Error</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>name does not exactly match a host profile or because the host profile has not yet been created.</td>
<td>name of the profile matches exactly, including the case of the letters.</td>
</tr>
<tr>
<td>WL2001</td>
<td>Text line is too many characters.</td>
<td>The line of text contains too many characters. Long lines of text cause the script to be hard to read or understand.</td>
<td>Break the line into multiple lines. Assign actions to variables instead of passing actions to other actions.</td>
</tr>
<tr>
<td>WL2002</td>
<td>Keyword Script already used.</td>
<td>There can be one keyword Script in each script; this error occurs when more than one Script keyword in the script.</td>
<td>Remove any extra Script keywords so that there is only one.</td>
</tr>
<tr>
<td>WL2003</td>
<td>Keyword Script must be the first statement.</td>
<td>The keyword Script must be the first statement in a script.</td>
<td>Move all statements before Script to after Script.</td>
</tr>
<tr>
<td>WL2004</td>
<td>Missing ( after keyword.</td>
<td>Almost all actions and keywords must have a ( after them. The ( is missing.</td>
<td>Add a ( after the keyword.</td>
</tr>
<tr>
<td>WL2005</td>
<td>Missing parameter for keyword or action.</td>
<td>The action or keyword requires one or more parameters, and there are no parameters present.</td>
<td>Add the correct number of parameters to the action or keyword.</td>
</tr>
<tr>
<td>WL2006</td>
<td>Second parameter for keyword must be TRUE or FALSE.</td>
<td>This error indicates an invalid parameter passed to an action or keyword. The most common cause of this error is an invalid parameter to Keypress_Key().</td>
<td>Carefully check the parameters passed to the action or keyword and study the error message to help fix the problem.</td>
</tr>
<tr>
<td>WL2007</td>
<td>Missing ) after parameter to</td>
<td>Most actions and keywords use ( and ). This action or keyword is missing the right parenthesis.</td>
<td>Add ), probably at the end of the line.</td>
</tr>
<tr>
<td>Error Number</td>
<td>Error Message</td>
<td>Description</td>
<td>Fixing the Error</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------</td>
<td>-------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>keyword. Missing ) after parameter to action.</td>
<td>Extra character(s) after parameter.</td>
<td>There are one or more extra characters after a parameter to an action or to a keyword.</td>
<td>Make sure that after a parameter, there is a right parenthesis ) or a comma and another parameter. Make sure that after the last ), there are no more characters.</td>
</tr>
<tr>
<td>WL2008</td>
<td>The variable name is already in use by a string variable.</td>
<td>Variable names may be declared once. This error occurs when a variable name is used in another variable declaration.</td>
<td>Change the name of the variable in the second declaration.</td>
</tr>
<tr>
<td>WL2009</td>
<td>The variable name is already in use by a number variable.</td>
<td>Variable names may be declared once. This error occurs when a variable name is used in another variable declaration.</td>
<td>Change the name of the variable in the second declaration.</td>
</tr>
<tr>
<td>WL2010</td>
<td>The variable name is already in use by a boolean variable.</td>
<td>Variable names may be declared once. This error occurs when a variable name is used in another variable declaration.</td>
<td>Change the name of the variable in the second declaration.</td>
</tr>
<tr>
<td>WL2011</td>
<td>Invalid activation method.</td>
<td>The method in the Activate action is not valid because it is not spelled correctly or because it's not a valid method.</td>
<td>Check the spelling. The method is case-sensitive. Make sure the method is in the list of activation methods.</td>
</tr>
<tr>
<td>WL2012</td>
<td>Non-printable character is not allowed.</td>
<td>A non-printable character is in the line of text. This can happen when pasting text from another program.</td>
<td>Re-type the line of text in the script editor.</td>
</tr>
<tr>
<td>WL2013</td>
<td>Blank characters are not allowed in the label. Blank characters are not allowed in</td>
<td>An invalid character is present in the script text.</td>
<td>Remove or change the character that the error message says is invalid.</td>
</tr>
<tr>
<td>Error Number</td>
<td>Error Message</td>
<td>Description</td>
<td>Fixing the Error</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>WL2015</td>
<td>Parameter cannot be empty. Not enough parameters; there must be 3 for On_Key.</td>
<td>Invalid character is not allowed. Invalid character is at the beginning of the line.</td>
<td>Add the missing parameter. All parameters must be on the same line of text.</td>
</tr>
<tr>
<td>WL2015</td>
<td>Parameter cannot be empty. Not enough parameters; there must be 3 for On_Key.</td>
<td>Parameter is missing from Activate, Keypress_Key, Label, Goto, or Call.</td>
<td></td>
</tr>
<tr>
<td>WL2016</td>
<td>Too many parameters.</td>
<td>The action contains too many parameters.</td>
<td>Check how many parameters the action needs and remove the extras.</td>
</tr>
<tr>
<td>WL2017</td>
<td>Invalid key value; must be 1-0xFFFF.</td>
<td>The key value for Activate(On_Key) cannot be zero.</td>
<td>Change the zero value to a valid key value.</td>
</tr>
<tr>
<td>WL2018</td>
<td>Invalid key modifier; must be None or Alt or Ctrl or Shift.</td>
<td>Activate(On_Key) contains a modifier that is not recognized or that is not supported.</td>
<td>Change the modifier to match exactly one of the supported modifiers; None or Alt or Ctrl or Shift.</td>
</tr>
<tr>
<td>WL2019</td>
<td>Missing : after keyword.</td>
<td>There must be a colon (:) after the keyword, before the label, for the actions Goto, Label, and Call.</td>
<td>Add a colon after the keyword, before the label.</td>
</tr>
<tr>
<td>WL2020</td>
<td>Extra</td>
<td>There are some characters</td>
<td>Remove all characters after the keyword, before the label.</td>
</tr>
<tr>
<td>Error Number</td>
<td>Error Message</td>
<td>Description</td>
<td>Fixing the Error</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>WL2021</td>
<td>The value actions are not allowed. The value Activate is a keyword, which is not allowed. No spaces or tabs are allowed in variable names. Unknown string variable.</td>
<td>A keyword or variable is being used for a variable name, or a variable name has spaces or Activate is using an unknown variable.</td>
<td>Use a variable name that is not an action or keyword. Make sure there are no spaces or tabs in the variable name. For Activate, define the variable before using Activate.</td>
</tr>
<tr>
<td>WL2022</td>
<td>Missing = after variable.</td>
<td>When setting a variable, use =.</td>
<td>Add a = after the variable.</td>
</tr>
<tr>
<td>WL2023</td>
<td>Missing assignment for variable.</td>
<td>After the = there must be a variable, action or constant, all on the same line.</td>
<td>If the assignment is split into two lines, combine them. If there is nothing after = add a variable or action or constant.</td>
</tr>
<tr>
<td>WL2024</td>
<td>Invalid assignment for variable.</td>
<td>A variable is being assigned to an unknown variable or to an unknown action or to an incorrect value.</td>
<td>If assigning to a variable or action, correct the case-sensitive spelling of the variable or action.</td>
</tr>
<tr>
<td>WL2025</td>
<td>Unknown action.</td>
<td>The script contains an action that is not known. Action names are case-sensitive.</td>
<td>Correct the spelling or the case of the action name. Make sure the _ is in the correct place in the name.</td>
</tr>
<tr>
<td>WL2026</td>
<td>Variable type is not the same as the action return type.</td>
<td>The return type of the action must be the same as the type of the variable.</td>
<td>Change the type of the variable or use another variable that has the same type as the action.</td>
</tr>
<tr>
<td>WL2027</td>
<td>Variable type is not the same as the variable type.</td>
<td>The type of the two variables must be the same.</td>
<td>Change the type of the variable or use another variable that has the same type.</td>
</tr>
<tr>
<td>Error Number</td>
<td>Error Message</td>
<td>Description</td>
<td>Fixing the Error</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>WL2028</td>
<td>Variable type is not the same as the assignment type.</td>
<td>The type of the two variables and the constant must be the same.</td>
<td>Change the type of the variable or constant. Use another variable or constant that has the same type.</td>
</tr>
<tr>
<td>WL2029</td>
<td>No closing quote is found in string. The parameter is not valid.</td>
<td>A string must be surrounded by quotation marks. A parameter must be a string, number, variable, or action.</td>
<td>If the string is missing a quotation mark, add one to the end of the string. If the parameter is not valid, check its spelling and the case of its letters. If the parameter is a variable, make sure the variable is declared above this line.</td>
</tr>
<tr>
<td>WL2030</td>
<td>The label cannot be an action. The label cannot be a variable. The parameter type doesn’t match the return type for the action. The parameter type must be the type.</td>
<td>Labels cannot be variables or actions. If an action is used as a parameter, the action return type must be the same as the parameter type.</td>
<td>For labels, use a unique name that is not an action or variable name. Instead of using an action as a parameter, use a variable that calls the action in a line before.</td>
</tr>
<tr>
<td>WL2031</td>
<td>Unexpected character.</td>
<td>The line of text is not quite right because a character is extra or out of place.</td>
<td>Use the error message information to help determine which character is incorrect.</td>
</tr>
<tr>
<td>WL2032</td>
<td>Not enough parameter values.</td>
<td>The action is missing one or more parameters.</td>
<td>Check the specification for the action and add parameters to it.</td>
</tr>
<tr>
<td>WL2033</td>
<td>Could not open file; error.</td>
<td>A script file could not be opened, probably because the user does not have the correct permission for the file or its folder.</td>
<td>Verify read and write permissions for the file and its folder.</td>
</tr>
<tr>
<td>WL2034</td>
<td>File is too small.</td>
<td>A script file must contain at least three bytes so that the script editor can determine if it’s Unicode, UTF-8, or ANSI</td>
<td>Use a different file that is larger.</td>
</tr>
<tr>
<td>Error Number</td>
<td>Error Message</td>
<td>Description</td>
<td>Fixing the Error</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>WL2035</td>
<td>File encoding not supported: Unicode big endian.</td>
<td>Script files may be encoded as Unicode (little endian), ANSI, or UTF-8. Script files cannot be encoded as Unicode big endian.</td>
<td>Open the file in Notepad, select <strong>File &gt; Save As &gt; Encoding</strong>, and change it to something other than Unicode big endian.</td>
</tr>
<tr>
<td>WL2036</td>
<td>Could not write to file.</td>
<td>A script file could not be written, probably because the user does not have the correct permission for the file or its folder.</td>
<td>Verify read and write permissions for the file and its folder.</td>
</tr>
<tr>
<td>WL2037</td>
<td>Memory allocation failed.</td>
<td>There is not enough memory left for the program to run correctly.</td>
<td>Restart the computer.</td>
</tr>
<tr>
<td>WL2038</td>
<td>Too many nested calls.</td>
<td>An action has a parameter that is an action that has a parameter that is an action and so on, too many times.</td>
<td>Make one action call per line of text, assigning the return value to a variable. Use the variable as a parameter to another action. Do not use actions as parameters.</td>
</tr>
<tr>
<td>WL2039</td>
<td>Script error.</td>
<td>The script contains a generic problem.</td>
<td>Use the error message to help determine the cause of the problem. Try commenting-out or removing some lines of the script to isolate the problem.</td>
</tr>
</tbody>
</table>
Wavelink Contact Information

If you have comments or questions regarding this product, please contact Wavelink Customer Service.

E-mail Wavelink Customer Support at: CustomerService@wavelink.com

For customers within North America and Canada, call the Wavelink Technical Support line at 801-316-9000 (option 2) or 888-699-9283.

For international customers, call the international Wavelink Technical Support line at +800 9283 5465.

For Europe, Middle East, and Africa, hours are 9 AM - 5 PM GMT.

For all other customers, hours are 7 AM - 7 PM MST.