HS3Scripting

User Manual

Created: Wednesday, July 03, 2013

HS3Scripting

Table of Contents

Scripting

In This Section

See Also

About Scripts

In This Section

See Also

Common Scripting Questions

How do you know when to use parenthesis when calling the scripting functions?

If the function is returning a value, you need to surround the parameters with parentheses. Otherwise, you need to omit them. Here is a function call that does not return a value:

```
hs.ExecX10 "A1","on"
```

The following function returns a status value:

```
if hs.IsOn("A1") then
```

Are the function names case-sensitive?

No. The function `hs.IsOn("A1")` and `hs.ison("A1")` are identical and work the same way.

Why are some functions prefaced with "hs." and some with "hsp."?

Functions that relate to the HomeSeer Phone begin with `hsp.`*x*, such as `hsp.GetLastVoiceCommand`. Functions that relate to HomeSeer itself begin with hs.*x*, such as `hs.ClearLog`.

See Also

Creating A Script
Debugging Scripts
Executing Single Script Statements
User Supported Scripts
VB.NET Scripts and NameSpaces

Creating A Script

All scripts must reside in the scripts folder in the HomeSeer application directory. Scripts commonly have the extension ".txt" or ".vbs" or ".vb" (for vb.net scripts) and are simple text files. You can edit your scripts in this directory. Here are the steps to create a simple script that turns a light off if it's on:

- In the *Events View*, click on the *add event* button. A new *Unnamed* event is created.
- Click on the event name to open the *Event Properties* dialog, give it a name like *light test.*
- Now click the *Actions* tab, select *Run Script* from the list of actions available to add to the event.
- Click the "*Switch To Advanced View*" button to expand the script action options.
- A dialog box displays asking you for the "*Existing or New Script Name*" of the script. Enter *lightoff.vbs*, click Ok. Note that for VBScript scripts, the extension for the script is ".vbs" or ".txt". For VB.NET scripts, use ".vb", for JScript enter the extension as ".js" and for Perlscript, enter the extension as ".pl".
- Click the "*Open Script Window*" button to display the script editor portion of the screen.
- The script already has the main subroutine defined for you. Modify the script so it looks like this:

```
Sub Main()
    if hs.ison("B2") then
        hs.execx10 "B2","off",0
    end if
End Sub
```

- Save the script by pressing the "*Save Script*" button, and then finish adding this action to the event by clicking the green "*Update*" button.
- Save the event by clicking the "*Save*" button, and you will be returned to the event list page where you can then test the event by clicking the green "*Run*" button shown next to the event name.
- VB.NET scripts use a slightly different format. The script is always passed a "parms" object which will be an array of objects that are the parameters. The object will be nothing if no parameters are supplied. The above script would be formatted as follows:

```
Sub Main(parms As Object)
If hs.IsOn("B2") then
    hs.ExecX10("B2","off",0)
End If
End Sub
```

- You can test script statements using the Control screen. Click on the Control button on the links bar, or the Tools button then the Control button, the Control screen appears. In the "Immediate Script Command" box, enter your script command. For example, to speak a phrase enter:

  hs.speak "hello"

  The system should speak. If you want to get the value of a HomeSeer device (for example, the current light value from a HSM100 sensor), first find the device code for the device. This is listed on the status page in the "Code" column. If you light device was code Q7, then enter this command in the Script Command box:

  hs.writelog "msg",hs.devicevalue("Q7")

  Check your event log, you should see an entry like:

  9/19/2009 12:45:18 PM - msg - 8900

  You can also display the message in a pop up message box by entering:

  msgbox hs.devicevalue("Q7")

See Also

Common Scripting Questions
Debugging Scripts
Executing Single Script Statements
User Supported Scripts
VB.NET Scripts and NameSpaces

Debugging Scripts

If your script has errors, the scripting engine will detect the error whenever the script is run. The error will appear in the event log. If your script is not working, check the log for errors. The log will contain the line number where the error occurred.

Do not allow your script to run for more than a few seconds. Scripts are to be used to perform a quick task that does not take a lot of time. The script engine will prompt you with a dialog box warning you that the script is taking a long time to run if the script is running for longer than 30 seconds. You can work around this, however. You can call the script function `hs.WaitEvents()` or `hs.WaitSecs()`. If you call this function within 30 seconds, the script will not time out. This will also let HomeSeer do other tasks while your script is executing.

See Also

Common Scripting Questions
Creating A Script
Executing Single Script Statements
User Supported Scripts
VB.NET Scripts and NameSpaces

Executing Single Script Statements

In the *Advanced* section of the Run Script action, you can add a single script statement. This allows you to execute script commands without creating a file. Statements are preceded with an ampersand (&) so HomeSeer knows to treat it as a statement. For example, the following *if then else logic* could be typed into the "OR Script Statement" field of the *Run Script* action:

```
&if hs.ison("b2") then hs.execx10 "b3","off",0 else hs.execx10 "b3","on",0
```

```
or
&hs.SetDeviceString("b2","Garage Open")
```

- Multiple statements may be added to the "OR Script Statement" field. Separate each statement with a colon (:).
- Only "hs" is supported as an object for script statements and not "ah".

See Also

Common Scripting Questions
Creating A Script
Debugging Scripts
User Supported Scripts
VB.NET Scripts and NameSpaces

User Supported Scripts

The HomeSeer message board has a section for scripts that are donated by users. There are a large variety of scripts available. Visit the message board at the HomeSeer website.

Script Languages

VBScript contains a large number of built-in commands. If you are new to scripting, get the VBScript documentation from Microsoft. A tutorial on VBScript is also available. You can get them off the Internet at: http://msdn.microsoft.com/scripting.

If you would rather use JScript (JavaScript) instead of VBScript (or any other supported script language) you can change the language HomeSeer uses by using a different extension with the script filename. The following scripting engines are supported:

- .txt and .vbs files are VBScript
- .js files are JavaScript
- .pl files are PerlScript

VB.NET Scripts

Scripts may also be written in VB.NET. Scripts written in this language are compiled then executed. The syntax of a VB.NET script follows the same rules as a standard VB.NET application. Consult the event log for compile errors when running your script.

VB.NET scripts differ from VBScript scripts significantly. In VB.NET, variables must be defined, and they must be given a variable type such as String, Integer, Short, Object, etc. Also, in VBScript scripts, subs are called without parenthesis, but functions require parenthesis, but in VB.NET this complication was removed and all parameters are enclosed in parenthesis. As an example:

```
hs.Speak "This is a test."
```

Is coded in VB.NET as:

```
hs.Speak("This is a test.")
```

Although learning to write VB.NET scripts is more like writing a program than a script, the speed advantages of compiled code may be worth it to you to learn how to do it. Microsoft does provide a (currently) free development environment called Visual Studio Express which may be used to author VB.NET scripts.

Because Visual Studio Express expects scripts to be a part of a class, and because your script source does not have the references to the HomeSeer system objects, you should create a class project in Visual Studio (call it whatever you like), then in the classname.vb module, type the following and insert your script code where indicated. This example uses a class name of Class1:

```
Public Class Class1
    Public hs As Object
    Public hssystem As Object
    Public hsp As Object

    '(Insert your Script Code Here)

    ' After editing, only copy/paste or save your script
    '   code to your HomeSeer VB.NET script file - do not
    '   save the Class1 constructor information or the
    '   public HomeSeer object variables.


End Class
```

Encrypted Scripts

HomeSeer scripts can be encrypted in two different ways. They can be encrypted using Microsoft's Script Encoder, or they can be encrypted using HomeSeer's built-in encryption. You can download Microsoft's script encoder from the Microsoft Script Technologies Downloads page.

- Microsoft's Script Encoder is not considered a secure encryption since decryption is done using public information, but it will stop the general public from viewing the code.

HomeSeer's encryption is done using the HomeSeer Script Encoder (HSSEncoder) tool which is included with the HomeSeer Developer Package (see the HomeSeer website).

Scripts that are encoded using Microsoft's Script Encoder are recognized by HomeSeer as scripts ending in .vbe for Visual Basic Encoded scripts, and .jse for JScript Encoded scripts.

Scripts that are encoded using HomeSeer's encryption are recognized by HomeSeer as scripts ending in .vbh for Visual Basic Scripts, .vben for VB.NET Scripts, and .jsh for JScript scripts.

HomeSeer encryption can also be applied to Active Server Pages (ASP) and their extension for the encrypted version of the page is .ash.

See Also

Common Scripting Questions
Creating A Script
Debugging Scripts
Executing Single Script Statements
VB.NET Scripts and NameSpaces

VB.NET Scripts and NameSpaces

.NET uses "Namespaces" to refer to large libraries of code, which are not included in your VB.NET program or script unless you tell it that you wish to have it included.

By default, VB.NET scripts executed by HomeSeer will have the System.dll referenced, which means that to use a namespace within that library, add the IMPORTS statement to the top of your VB.NET script like this:

```
IMPORTS System.IO
IMPORTS System.Net
```

- Note that all namespaces under System are not necessarily included in System.dll - some System namespaces are in additional dll files.

If you wish to use other namespaces referenced in other libraries, you must first tell HomeSeer to include the library reference when it initializes the script engine. To do this, add your reference to the INI entry "ScriptingReferences", which is under the [Settings] section in your \Config\Settings.INI file.

You can add references by including the namespace, a semicolon, and the dll file name in this entry. Multiple references can be added by separating them with a comma. Here is an example that adds a reference to 2 namespaces:

```
ScriptingReferences =

    System.Management;System.Management.dll,System.Drawing;System.Drawing.dll
```

When broken down into individual parts:

```
ScriptingReferences =
```

```
System.Management;System.Management.dll,System.Drawing;System.Drawing.dll
  (Namespace1)    (DLL for Namespace1)  (Namespace2)  (DLL for Namespace2)
```

See Also

- Common Scripting Questions
- Creating A Script
- Debugging Scripts
- Executing Single Script Statements
- User Supported Scripts

Applications and Plugins

In This Section

- System Information
- System Functions
- INI File Editing
- Plug-Ins
- Logging
- Web Pages
- Callbacks
- Launch
- SendMessage
- ReplaceVariables

See Also

- About Scripts
- Computer
- Devices
- Email
- Events
- Internet
- Phone
- Scripts
- Speech Recognition
- Strings, Global Variables, and Encryption
- Time and Calendar
- Text-To-Speech and Media

System Information

In This Section

- AppStarting
- DebugMode
- GetAppPath
- InterfaceVersion
- IsLicensed
- ShuttingDown
- SystemUptime
- SystemUpTimeTS
- Version

See Also

System Functions
INI File Editing
Plug-Ins
Logging
Web Pages
Callbacks
Launch
SendMessage
ReplaceVariables

AppStarting

Purpose

Indicates if the HomeSeer application is currently starting or doing startup processing.

Parameters

None.

Returns

Return value: **status**
Type: **boolean**
Description: Returns **True** if the application is starting or doing startup processing.

See Also

DebugMode
GetAppPath
InterfaceVersion
IsLicensed
ShuttingDown
SystemUptime
SystemUpTimeTS
Version

DebugMode

Purpose

This command sets or disables HomeSeer's built-in debugging trace functions. When enabled, debug information is written to the "\Debug Logs" directory into a separate file for each debug type enabled. A "Debug_Composite.log" file in the debug logs directory contains all debug information written in all logs, and the "Debug_Other.log" contains debugging information not categorized into one of the debug types listed below as well as the HomeSeer log entries produced during the time when debug logging is enabled. Writing debug information can add to the overall system load, so please use this under advisement from HomeSeer Technologies support personnel.

Parameters

Parameter: **mode**
Type: **long (.NET Integer)**
Description: The debug mode for HomeSeer to be set to (see below). Multiple modes can be logically OR'd together to enable several debug modes at once.

Returns

None.

Debug Mode Values

Value	Function Trace
0	Disable debugging (normal)
1	Script activity
2	Event activity
4	Condition checks
8	Device value changes
16	HomeSeer procedure calls
128	Plug-In Procedure calls
256	Database Procedure Debugging
512	Web/E-Mail Server Activity

Example

Turn on debugging for script, event, and plug-ins: (1 + 2 + 128 = 131)

```
Sub Main()
	hs.DebugMode = 131
End Sub
```

See Also

AppStarting
GetAppPath
InterfaceVersion
IsLicensed
ShuttingDown
SystemUptime
SystemUpTimeTS
Version

GetAppPath

Purpose

Returns the path to the HomeSeer installation director. This is useful for finding HomeSeer-specific files like the event log and script-created files.

Parameters

None.

Returns

Return value: **application path**

Type: **string**
Description: The path returned is not terminated by a directory path separator. Use of GetAppPath to construct a valid path will require the addition of "/" before additional path names or files are added.

Example

```
sub main()

      dim s

      s = hs.GetAppPath
      msgbox "The HomeSeer path is: " & s

end sub
```

See Also

AppStarting
DebugMode
InterfaceVersion
IsLicensed
ShuttingDown
SystemUptime
SystemUpTimeTS
Version

InterfaceVersion

Purpose

Returns the current version of HomeSeer Plug-In API Interface. This procedure is called by plug-ins to determine the capability level of the interface it is working with.

Parameters

None.

Returns

Return value: **version**
Type: **integer (.NET Short)**
Description: For HS2, the returned value is 3, and for HS3, the returned value is 4.

Example

```
Sub Main(ByVal Parm As Object)

          hs.WriteLog("Info","The API interface version of HomeSeer is " &
hs.InterfaceVersion.ToString)

End Sub
```

See Also

AppStarting
DebugMode
GetAppPath
IsLicensed
ShuttingDown
SystemUptime
SystemUpTimeTS
Version

IsLicensed

Purpose

Returns True/False indicating whether HomeSeer has been fully licensed.

Parameters

None.

Returns

Return value: **license status**
Type: **boolean**

Example

```
sub main()

     If hs.IsLicensed then
         hs.WriteLog "Info","HomeSeer is licensed, thank you."
     else
      hs.WriteLog "Info","This copy of HomeSeer is not currently licensed."
     end if

end sub
```

See Also

AppStarting
DebugMode
GetAppPath
InterfaceVersion
ShuttingDown
SystemUptime
SystemUpTimeTS
Version

ShuttingDown

Purpose

This allows applications and plug-ins to determine if HomeSeer has started the shutdown process.

Parameters

None.

Returns

Return value: **Shut Down Status**
Type: **Boolean**
Description: If TRUE, the system is in the process of shutting down.

See Also

AppStarting
DebugMode
GetAppPath
InterfaceVersion
IsLicensed
SystemUptime
SystemUpTimeTS
Version

SystemUptime

Purpose

Returns the amount of time HomeSeer has been running. Time is displayed in the format `days hours:minutes:seconds`.

Parameters

None.

Returns

Return value: **time**
Type: **string**

Example

```
' Set a virtual device to display the system uptime
Sub Main(ByVal Parms As Object)
  hs.SetDeviceString(1234, "Uptime: " & hs.SystemUpTime, True)
End Sub
' the display might be: Uptime: 1 Days 12:23:07
```

See Also

AppStarting
DebugMode
GetAppPath
InterfaceVersion
IsLicensed
ShuttingDown
SystemUpTimeTS
Version

SystemUpTimeTS

Purpose

Returns the amount of time HomeSeer has been running in a TimeSpan structure.

Parameters

None.

Returns

Return value: **time**
Type: **Timespan**

Example

```
Sub Main(ByVal Parms As Object)

    Dim TS As TimeSpan
    TS = hs.SystemUpTimeTS
    hs.WriteLog("Up Time", "HomeSeer has been running for " & TS.Days.ToString & " days, " & _
                           TS.Hours.ToString & " hours, and " & TS.Minutes.ToString & " minutes.")

End Sub
```

See Also

AppStarting
DebugMode
GetAppPath
InterfaceVersion
IsLicensed
ShuttingDown
SystemUptime
Version

Version

Purpose

Returns the current version of HomeSeer.

Parameters

None.

Returns

Return value: **version string**
Type: **string**

Example

```
Sub Main(ByVal Parms As Object)

    hs.WriteLog("Version Info", "HomeSeer HS3 is currently version " & hs.version)

End Sub
```

See Also

AppStarting
DebugMode
GetAppPath
InterfaceVersion
IsLicensed
ShuttingDown
SystemUptime
SystemUpTimeTS

System Functions

Purpose

System functions are used when a function must interface with either the HomeSeer application or the HomeSeer Phone application. Since voice recognition and text-to-speech will use different output and input devices, these functions handle routing the sound information to and from the devices.

When a script calls `hs.Speak`, the sound is normally sent directly to the sound card on the PC. However, if the script is run in response to a voice command over the phone, it is normally desired to send the sound over the phone's handset. By calling `system.Speak`, HomeSeer knows where the request originated, and therefore routes the sound out through the proper device.

The system functions listed here are merely wrappers for the real functions in either the `hs` (HomeSeer object) or `hsp` (HomeSeer Phone object).

- When using the system functions, note that they only work from within the script that was initially launched. If you call a second script (using `hs.Run` or `hs.RunEx`), any system functions used in these scripts will not work on the desired audio channel. This is due to the way HomeSeer creates the system object when the initial script is launched. It has no information about sub scripts as to where the script was launched from. It therefore creates a default system object that assumes the script is dealing with the default audio channel.
- **Due to .NET using "system" as a NameSpace, VB.NET scripts must use "hssystem" in place of "system".**

Example

Speak a phrase

```
system.speak text as string, optional wait as boolean
```

Add a voice command

```
system.AddVoiceCommand command as string
```

Get the last recognized voice command

```
system.LastVoiceCommand as string
```

Clear all voice commands that were set using AddVoiceCommand

If the script is used over the phone and the "AddVoiceCommand" was used, then this must be called when the script exists so that the standard voice commands can be re-enabled.

```
system.ClearAllVoiceCommands
```

Stop voice recognition

```
system.StopListen
```

Start voice recognition

```
system.StartListen
```

Get the phone line that triggered the event (0=triggered by HomeSeer, and not the phone)

```
system.LastLine or system.SpeakSource
```

Start voice recognition

```
system.StartListen
```

See Also

System Information
INI File Editing
Plug-Ins
Logging
Web Pages
Callbacks
Launch
SendMessage
ReplaceVariables

BackupDB

Purpose

This command will close the currently active HomeSeer configuration database and will make a backup copy of it in the Backup directory under the Config directory of HomeSeer.

Parameters

None.

Returns

Parameter: **result**
Type: **string**
Description: The result of the backup operation. If it is an empty string ("") then the operation was successful. If the result is not empty, then it will contain information regarding the error or problem encountered during the backup procedure.

Notes / Additional Information

The default number of backup copies of the database varies with the HomeSeer edition, but the typical amount is 10 for the standard edition of HomeSeer. The backup copies are numbered from 1 to the highest number of backups that are retained. The system overwrites the oldest file when this command is issued. Because the rotation is based upon file date/time, the 10th numbered file (in the case of a 10 limit rotation) is not always the oldest backup. You must examine the file modification dates/times to determine which backup is the most recent or oldest.

You can control the number of copies that are retained by adding the following setting to your settings.ini file in the CONFIG directory:

- Edit this file when HomeSeer is shut down.

[Database]
Backup_Copies=15

In the above example, 15 copies will be retained. The number of copies to be retained must be greater than 0 and less than or equal to 50 or the value will be reset to its default value.

See Also

PowerFailRecover
ScheduleFile
Shutdown
System

PowerFailRecover

Purpose

This command will trigger a power failure recovery operation similar to the one that is started automatically if it is enabled in the system configuration.

The recovery takes place using the number of hours set in the power fail options, and it starts from the time an event was last run successfully on the system.

Parameters

None.

Returns

None.

See Also

BackupDB
ScheduleFile
Shutdown
System

ScheduleFile

Purpose

This property can be set and read. Setting this property configures HomeSeer to use a new configuration file. Reading this property reports the currently configured configuration file. Configuration files hold all configured devices and events.

Parameters

Parameter: **Filename**
Type: **String**
Description: The complete path to a new configuration file to use, or the filename only if the configuration file is already stored in the HomeSeer\Config directory.

Returns

Return value: **Filename**
Type: **String**
Description: Returns the full path and name of the current configuration file.

Example

```
' Set a new configuration file
hs.ScheduleFile = "c:\NewConfig.hsd"

' Read the current configuration file
Dim Config_File As String
Config_File = hs.ScheduleFile
```

See Also

BackupDB
PowerFailRecover
Shutdown
System

Shutdown

Purpose

Causes HomeSeer to shut down immediately. This has the same affect as selecting `File` > `Exit` from the file menu within HomeSeer. If HomeSeer Phone is running, that will be shut down also.

Parameters

None.

Returns

None.

See Also

BackupDB
PowerFailRecover
ScheduleFile
System

System

Purpose

Scripts can access the system object directly without using this function. However, external programs that wish to access the system object need to call this function to get access to it. The system object is an independent interface that allows for access to either the HS (HomeSeer) or HSP (HomeSeer Phone) object.

Parameters

None.

Returns

Return value: **system object**
Type: **object**
Description: Returns a reference to the system object.

Example

```
dim system

set system = hs.system
```

See Also

BackupDB
PowerFailRecover
ScheduleFile
Shutdown

INI File Editing

In This Section

ClearINISection
GetINISection
GetINISectionEx
GetINISetting
SaveINISetting

See Also

ClearINISection

Purpose

Clears an entire section in an INI file.

- HomeSeer will be reset to its default settings if the "Settings" section is cleared in the `settings.ini` file.

Parameters

Parameter: **section**
Type: **string**
Description: Name of the section in the INI to be cleared, such as "Settings" for the HomeSeer settings section.

Parameter: **filename**
Type: **string**
Description: Name of the INI file to be accessed. For instance, the INI file for HomeSeer is `settings.ini`.

- The file name is relative to the "config" folder in the HomeSeer program directory (`C:\Program Files\HomeSeer 2\Config` by default).

Returns

None.

See Also

GetINISection
GetINISectionEx
GetINISetting
SaveINISetting

GetINISection

Purpose

Returns all values in the given INI section. Each entry in the section is separated with a NULL character.

Parameters

Parameter: **section**
Type: **string**
Description: Name of the section in the INI file to get, like "Settings" for the HomeSeer settings section.

Parameter: **filename**
Type: **string**
Description: File name of the INI file to access, such as "settings.ini". The file name is relative to the "config" folder in the HomeSeer program directory (`C:\Program Files\HomeSeer 2\Config` by default).

Returns

Return value: **ini section**
Type: **string**

Example

```
Sub Main(ByVal Parms As Object)
    Dim Items() As String
    Dim Section As String

    Section = hs.GetINISection("Settings", "settings.ini")
    Items = Section.Split(Chr(0))
    If Items IsNot Nothing AndAlso Items.Count > 0 Then
        For Each s As String In Items
            If String.IsNullOrEmpty(s) Then Continue For
            hs.WriteLog("Items", s)
        Next
    End If

End Sub
```

See Also

ClearINISection
GetINISectionEx
GetINISetting
SaveINISetting

GetINISectionEx

Purpose

Returns all values in the given INI section. Each entry in the section is an element in a string array.

Parameters

Parameter: **section**
Type: **string**
Description: Name of the section in the INI file to get, like "Settings" for the HomeSeer settings section.

Parameter: **filename**
Type: **string**
Description: File name of the INI file to access, such as "settings.ini". The file name is relative to the "config" folder in the HomeSeer program directory (`C:\Program Files\HomeSeer 2\Config` by default).

Returns

Return value: **ini section**
Type: **string array**

Example

```
Sub Main(ByVal Parms As Object)
    Dim Items() As String

    Items = hs.GetINISection("Settings", "settings.ini")
    If Items IsNot Nothing AndAlso Items.Count > 0 Then
        For Each s As String In Items
            If String.IsNullOrEmpty(s) Then Continue For
            hs.WriteLog("Items", s)
        Next
```

```
    End If

End Sub
```

See Also

ClearINISection
GetINISection
GetINISetting
SaveINISetting

GetINISetting

Purpose

Returns the value associated with the requested key from an INI file.

Parameters

Parameter: **section**
Type: **string**
Description: Name of section in INI file to get,such as "Settings" for the HomeSeer settings section.

Parameter: **key**
Type: **string**
Description: Name of the key in the INI file to access.

Parameter: **default**
Type: **string**
Description: The default value to return if the key is not found.

Parameter: **filename**
Type: **string** (optional)
Description: The file name of the INI file to access. The file name is relative to the "config" folder in the HomeSeer program directory (`C:\Program Files\HomeSeer 2\Config` by default). If this parameter is omitted, the HomeSeer `settings.ini` file is used.

Returns

Return value: **ini key value**
Type: **string**

Example

```
Sub Main(ByVal Parms As Object)
    Dim ConfigFileName As String = ""

    ' Get the name of the current HomeSeer configuration file.
    ConfigFileName = hs.GetINISetting("Settings", "configfile", "")
    hs.WriteLog("Config", "The current HomeSeer configuration file is " & ConfigFileName)

End Sub
```

See Also

ClearINISection
GetINISection
GetINISectionEx
SaveINISetting

SaveINISetting

Purpose

Saves a key/value pair in an INI file.

Parameters

Parameter: **section**
Type: **string**
Description: Name of the section in the INI file to save to, like "Settings" for the HomeSeer settings section.

Parameter: **key**
Type: **string**
Description: Name of the key in the INI file to access.

Parameter: **value**
Type: **variant**
Description: The value to save in the given key.

Parameter: **filename**
Type: **string** (optional)
Description: This is the file name of the INI file to access. The file name is relative to the "config" folder in the HomeSeer program directory (`C:\Program Files\HomeSeer 2\Config` by default). If this parameter is omitted, the HomeSeer `settings.ini` file is used.

Returns

None.

Example

```
hs.SaveINISetting("My Settings", "Zip Code", "49601", "My_App_Settings.ini")
```

See Also

ClearINISection
GetINISection
GetINISectionEx
GetINISetting

Plug-Ins

In This Section

GetHSPRef
GetPluginsList
RegisterLinkEx

See Also

System Information
System Functions
INI File Editing
Logging
Web Pages
Callbacks
Launch
SendMessage
ReplaceVariables

GetHSPRef

Purpose

This returns a reference to the HomeSeer Phone interface. This function is used primarily for plug-ins to communicate with the HomeSeer Phone directly without having to build a reference to it that might cause HomeSeer Phone to run when the user did not want it to run.

Parameters

None.

Returns

Return value: **object reference**
Type: **object**

See Also

GetPluginsList
RegisterLinkEx

GetPluginsList

Purpose

Returns an array of all plug-in names for plug-ins that are enabled.

Parameters

None.

Returns

Return value: **plug-in array**
Type: **Array of String**

Note

If a plug-in has instance information, then the instance name is appended to the plug-in name, with a colon (:) separating them. It should therefore be noted that this procedure may return the same plug-in name multiple times if there are multiple instances of that plug-in loaded.

Example

```
Sub Main(ByVal Parms As Object)
    Dim List() As String

    List = hs.GetPluginsList
    If List IsNot Nothing AndAlso List.Count > 0 Then
        For Each P As String In List
            If String.IsNullOrEmpty(P) Then Continue For
            hs.WriteLog("Plug-In List", P & " is currently enabled.")
        Next
    End If
```

```
End Sub
```

See Also

GetHSPRef
RegisterLinkEx

RegisterLinkEx

Purpose

This procedure registers a web page link with HomeSeer that is handled by a plug-in.

- This procedure only works with a static object reference such as a plug-in - it cannot be used by scripts.

Parameters

Parameter: **object ref**
Type: **code object**
Description: The object reference provided here is a code object such as a form or class, that contains the BuildPage, PagePut, etc. procedures that HomeSeer will call to provide the web page functionality.

Parameter: **plug-in name**
Type: **string**
Description: Name of the plug-in or program that this link is associated with. HomeSeer uses this to keep the links updated in the event that the plug-in or program is removed or otherwise goes away.

Returns

None.

Example

```
hs.RegisterLinkEx FrmWebPage, IFACE_NAME
```

See Also

GetHSPRef
GetPluginsList

Logging

In This Section

ClearLog
GetLog
LogGet
NoLog
WriteLog
WriteLogEx
WriteLogDetail

See Also

System Information
System Functions
INI File Editing
Plug-Ins
Web Pages
Callbacks
Launch
SendMessage
ReplaceVariables

ClearLog

Purpose

Clears the event log in memory. The event log file specified in the General Setup screen will not be touched.

Parameters

None.

Returns

None.

Example

```
hs.ClearLog
```

See Also

GetLog
LogGet
NoLog
WriteLog
WriteLogEx
WriteLogDetail

GetLog

In This Section

LogEntry Structure
GetLog_FullFilter
GetLog_Date
GetLog_Date_Text
GetLog_Date_Priority
GetLog_Date_ErrorCode

See Also

ClearLog
LogGet
NoLog
WriteLog
WriteLogEx
WriteLogDetail

LogEntry Structure

This structure is used by the GetLog_ functions listed in this section, and is as follows:

```
Public Structure LogEntry
    Public LogTime As Date  ' The date and time the log entry was recorded.
    Public LogType As String  ' The 'type' string as logged.
    Public LogText As String  ' The main message text of the log entry.
    Public LogStyleColor As String ' The color code string associated with this log entry.
    Public LogPriority As Integer ' The priority (0 = None Specified)
    Public LogFrom As String  ' Information on the software that generated the log entry.
    Public LogErrorCode As Integer ' An error code as provided by the originator of the log entry.
    Public LogLength As Integer ' The length of the full text of the log entry.
End Structure
```

See Also

GetLog_FullFilter
GetLog_Date
GetLog_Date_Text
GetLog_Date_Priority
GetLog_Date_ErrorCode

GetLog_FullFilter

```
Function GetLog_FullFilter(ByVal StartDate As Date, ByVal EndDate As Date, _
                           ByVal mType As String, ByVal mEntry As String, ByVal mEntry_RegEx As Boolean, _
                           ByVal Pri_Start As Integer, ByVal Pri_End As Integer, ByVal Show_NoPri As Boolean, _
                           ByVal ErrorCode As Integer, ByVal ShowAllErrorCode As Boolean) _
        As LogEntry()
```

Purpose

This function retrieves entries from the system log database and returns them as an array of LogEntry structures. This function provides the most filter parameters possible for selection of log entries. Use this function when you need a very precise set of log entries.

Parameters

Parameter: **StartDate**
Type: **Date/Time**
Description: This is the starting date for the log entries that you want retrieved. If you do not care about a starting date, use Date.MinValue

Parameter: **EndDate**
Type: **Date/Time**
Description: This is the ending (latest) date for the log entries that you want retrieved. If you do not care about an ending date, use Date.MaxValue

Parameter: **mType**
Type: **String**

Description: This is the log entry type, such as "Info" or "Error" - these must EXACTLY match what you are searching for - leave empty to not use this.

Parameter: **mEntry**
Type: **String**
Description: This is the entry text to find. This is an exact match field unless wildcards or RegEx (next parameter) is used. To use a wildcard, use the percent (%) character. For example, to match everything that starts with "Super", use "Super%" which will match SuperDuper, Super Cool, and Super Delicious.

Parameter: **mEntry_RegEx**
Type: **Boolean**
Description: When this parameter is TRUE, the previous parameter (mEntry) contains a Regular Expression to be run on the log message field retrieved from the database. For help with Regular Expressions, see Regular-Expressions.info

Parameter: **Pri_Start**
Type: **Integer**
Description: This is the starting priority for log entries to be retrieved. If you wish to retrieve log entries which are priority 1 through 5, provide a value of 1 here and a value of 5 in the Pri_End parameter. If you do not need to filter by priority, use the value -1.

Parameter: **Pri_End**
Type: **Integer**
Description: This is the ending priority value for the log entries to be retrieved. If you wish to retrieve log entries which are priority 1 through 5, provide a value of 1 in the Pri_Start parameter and a value of 5 in this parameter. If you do not need to filter by priority, use the value -1.

Parameter: **Show_NoPri**
Type: **Boolean**
Description: When set to True, unprioritized entries (Priority = 0) are included in the selection in addition to the priorities selected with Pri_Start and Pri_End.

Parameter: **ErrorCode**
Type: **Integer**
Description: This is the error code to select records with. Use a value of -1 (Or use ShowAllErrorCode) if you do not care to filter log entries using the Error Code value.

Parameter: **ShowAllErrorCode**
Type: **Boolean**
Description: When set to True, the ErrorCode parameter is ignored and all log entries that match the other filters are returned.

Returns

Return value: **LogEntry**
Type: **Array of Structure LogEntry**
Description: See LogEntry Structure.

Example

This example retrieves all log entries from a week ago to today, with a type of "Error", and a priority between 1 and 3 inclusive or unprioritized (Priority = 0):

```
Dim Logs() As HomeSeerAPI.LogEntry
Logs = hs.GetLog_FullFilter(Now.AddDays(-7), Now, "Error", "", False, 1, 3, True, -1, True)
If Logs IsNot Nothing AndAlso Logs.Count > 0 Then
   hs.WriteLog("Info", Logs.Count.ToString & " log entries retrieved, and this just added another!")
End If
```

See Also

LogEntry Structure
GetLog_Date
GetLog_Date_Text
GetLog_Date_Priority
GetLog_Date_ErrorCode

GetLog_Date

```
Function GetLog_Date(ByVal StartDate As Date, ByVal EndDate As Date) As LogEntry()
```

Purpose

This function retrieves entries from the system log database and returns them as an array of LogEntry structures. Use this function when you need to only filter log entries by date.

Parameters

Parameter: **StartDate**
Type: **Date/Time**
Description: This is the starting date for the log entries that you want retrieved. If you do not care about a starting date, use Date.MinValue

Parameter: **EndDate**
Type: **Date/Time**
Description: This is the ending (latest) date for the log entries that you want retrieved. If you do not care about an ending date, use Date.MaxValue

Returns

Return value: **LogEntry**
Type: **Array of Structure LogEntry**
Description: See LogEntry Structure.

Example

This example retrieves all log entries from a week ago to today:

```
Dim Logs() As HomeSeerAPI.LogEntry
Logs = hs.GetLog_Date(Now.AddDays(-7), Now)
If Logs IsNot Nothing AndAlso Logs.Count > 0 Then
   hs.WriteLog("Info", Logs.Count.ToString & " log entries retrieved, and this just added another!")
End If
```

See Also

LogEntry Structure
GetLog_FullFilter
GetLog_Date_Text
GetLog_Date_Priority
GetLog_Date_ErrorCode

GetLog_Date_Text

```
Function GetLog_Date_Text(ByVal StartDate As Date, ByVal EndDate As Date, _
                          ByVal mType As String, ByVal mEntry As String, ByVal mEntry_RegEx As Boolean) _
        As LogEntry()
```

Purpose

This function retrieves entries from the system log database and returns them as an array of LogEntry structures. This function provides the ability to filter on date and the log entry text fields of type and message. Use this function when you are looking for specific log entries without knowing the priority or error codes.

Parameters

Parameter: **StartDate**
Type: **Date/Time**
Description: This is the starting date for the log entries that you want retrieved. If you do not care about a starting date, use Date.MinValue

Parameter: **EndDate**
Type: **Date/Time**
Description: This is the ending (latest) date for the log entries that you want retrieved. If you do not care about an ending date, use Date.MaxValue

Parameter: **mType**
Type: **String**

Description: This is the log entry type, such as "Info" or "Error" - these must EXACTLY match what you are searching for - leave empty to not use this.

Parameter: **mEntry**
Type: **String**
Description: This is the entry text to find. This is an exact match field unless wildcards or RegEx (next parameter) is used. To use a wildcard, use the percent (%) character. For example, to match everything that starts with "Super", use "Super%" which will match SuperDuper, Super Cool, and Super Delicious.

Parameter: **mEntry_RegEx**
Type: **Boolean**
Description: When this parameter is TRUE, the previous parameter (mEntry) contains a Regular Expression to be run on the log message field retrieved from the database. For help with Regular Expressions, see Regular-Expressions.info

Returns

Return value: **LogEntry**
Type: **Array of Structure LogEntry**
Description: See LogEntry Structure.

Example

This example retrieves all log entries from a week ago to today, with a type of "Error":

```
Dim Logs() As HomeSeerAPI.LogEntry
Logs = hs.GetLog_Date_Text(Now.AddDays(-7), Now, "Error", "", False)
If Logs IsNot Nothing AndAlso Logs.Count > 0 Then
   hs.WriteLog("Info", Logs.Count.ToString & " log entries retrieved, and this just added another!")
End If
```

See Also

LogEntry Structure
GetLog_FullFilter
GetLog_Date
GetLog_Date_Priority
GetLog_Date_ErrorCode

GetLog_Date_Priority

```
Function GetLog_Date_Priority(ByVal StartDate As Date, ByVal EndDate As Date, _
ByVal Pri_Start As Integer, ByVal Pri_End As Integer, ByVal Show_NoPri As Boolean) _
        As LogEntry()
```

Purpose

This function retrieves entries from the system log database and returns them as an array of LogEntry structures. This function provides filtering the entries using the date and priority.

Parameters

Parameter: **StartDate**
Type: **Date/Time**
Description: This is the starting date for the log entries that you want retrieved. If you do not care about a starting date, use Date.MinValue

Parameter: **EndDate**
Type: **Date/Time**
Description: This is the ending (latest) date for the log entries that you want retrieved. If you do not care about an ending date, use Date.MaxValue

Parameter: **Pri_Start**
Type: **Integer**
Description: This is the starting priority for log entries to be retrieved. If you wish to retrieve log entries which are priority 1 through 5, provide a value of 1 here and a value of 5 in the Pri_End parameter. If you do not need to filter by priority, use the value -1.

Parameter: **Pri_End**
Type: **Integer**
Description: This is the ending priority value for the log entries to be retrieved. If you wish to retrieve log entries which are priority 1 through 5, provide a value of 1 in the Pri_Start parameter and a value of 5 in this parameter. If you do not need to filter by priority, use the value -1.

Parameter: **Show_NoPri**
Type: **Boolean**
Description: When set to True, unprioritized entries (Priority = 0) are included in the selection in addition to the priorities selected with Pri_Start and Pri_End.

Returns

Return value: **LogEntry**
Type: **Array of Structure LogEntry**
Description: See LogEntry Structure.

Example

This example retrieves all log entries from a week ago to today, with a priority between 1 and 3 inclusive or unprioritized (Priority = 0):

```
Dim Logs() As HomeSeerAPI.LogEntry
Logs = hs.GetLog_Date_Priority(Now.AddDays(-7), Now, 1, 3, True)
If Logs IsNot Nothing AndAlso Logs.Count > 0 Then
   hs.WriteLog("Info", Logs.Count.ToString & " log entries retrieved, and this just added another!")
End If
```

See Also

LogEntry Structure
GetLog_FullFilter
GetLog_Date
GetLog_Date_Text
GetLog_Date_ErrorCode

GetLog_Date_ErrorCode

```
Function GetLog_Date_ErrorCode(ByVal StartDate As Date, ByVal EndDate As Date, _
                               ByVal ErrorCode As Integer, ByVal ShowAllErrorCode As Boolean) _
        As LogEntry()
```

Purpose

This function retrieves entries from the system log database and returns them as an array of LogEntry structures. This function provides filtering of the log entries by date, and the error code associated with the log entry.

Parameters

Parameter: **StartDate**
Type: **Date/Time**
Description: This is the starting date for the log entries that you want retrieved. If you do not care about a starting date, use Date.MinValue

Parameter: **EndDate**
Type: **Date/Time**
Description: This is the ending (latest) date for the log entries that you want retrieved. If you do not care about an ending date, use Date.MaxValue

Parameter: **ErrorCode**
Type: **Integer**
Description: This is the error code to select records with. Use a value of -1 (Or use ShowAllErrorCode) if you do not care to filter log entries using the Error Code value. Error codes are provided by the procedure that added the log entry - they are not standardized. Consult the author of 3rd party provided scripts and plug-ins to obtain a list of error codes that they may have used.

Parameter: **ShowAllErrorCode**
Type: **Boolean**
Description: When set to True, the ErrorCode parameter is ignored and all log entries that match the other filters are returned.

Returns

Return value: **LogEntry**
Type: **Array of Structure LogEntry**
Description: See LogEntry Structure.

Example

This example retrieves all log entries from a week ago to today, with an error code of 4166:

```
Dim Logs() As HomeSeerAPI.LogEntry
Logs = hs.GetLog_Date_ErrorCode(Now.AddDays(-7), Now, 4166, False)
If Logs IsNot Nothing AndAlso Logs.Count > 0 Then
   hs.WriteLog("Info", Logs.Count.ToString & " log entries retrieved, and this just added another!")
End If
```

See Also

LogEntry Structure
GetLog_FullFilter
GetLog_Date
GetLog_Date_Text
GetLog_Date_Priority

LogGet

Purpose

Retrieves the current HomeSeer log buffer contents.

Parameters

Parameters: none

Returns

Return value: **buffer**
Type: **string**
Description: The contents of the HomeSeer log buffer.

The HomeSeer log is written to the HomeSeer log file (typically HomeSeer.log) and is stored in memory, up to the limit the user has set on the "General" tab of the HomeSeer configuration pages. The buffer returned here contains the log entries up to that limit, or since the log buffer in memory was last cleared by the user or a script action.

See Also

ClearLog
GetLog
NoLog
WriteLog
WriteLogEx
WriteLogDetail

NoLog

```
Property NoLog() As Boolean
```

Purpose

This property allows you to get or set whether logging is to take place on the process this property is set from. If you set this property in a script, then all logging entries from procedures run from the script are stopped. If this property is set from a plug-in, then logging from that plug-in is prevented until NoLog is reset or the plug-in is shut-down and restarted.

Note: Log entries (e.g. WriteLogDetail) which include a Priority value of 1 (highest) are always written regardless of the NoLog setting.

Parameters

Parameter: **NoLog**
Type: **Boolean**
Description: When set to True, logging for the process thread is turned off.

Returns

Return value: **NoLog**
Type: **Boolean**
Description: Provides the current NoLog setting for the process the property is retrieved from.

Example

To disable logging for the current process thread (script, plug-in, event):

```
hs.NoLog = True
```

See Also

ClearLog
GetLog
LogGet
WriteLog
WriteLogEx
WriteLogDetail

WriteLog

Purpose

Writes a message to the event log.

Parameters

Parameter: **type**
Type: **string**
Description: This is a string that defines the type of event like "Error" or "Info". It can be anything you like. Common message types are "Info","Warning", and "Error".

Parameter: **message**
Type: **string**
Description: This is the text to be displayed in the log, like a descriptive error message.

Returns

None.

Example

```
Sub Main()

    hs.WriteLog "Error", "An error has occurred in my script!"


End Sub
```

See Also

ClearLog
GetLog
LogGet
NoLog
WriteLogEx
WriteLogDetail

WriteLogEx

Purpose

Writes a message to the event log, with an optional COLOR specified.

Parameters

Parameter: **type**
Type: **string**
Description: This is a string that defines the type of event like "Error" or "Info". It can be anything you like. Common message types are "Info","Warning", and "Error".

Parameter: **message**
Type: **string**
Description: This is the text to be displayed in the log, like a descriptive error message.

Optional Parameter: **color**
Type: **string**
Description: This is the color code that you want associated with the log entry when you view it in the web browser. The color code must be in the #XXXXXX format, which is the # symbol followed by hexadecimal values for Red, Green, and Blue. Here are some examples of colors and their color values:

WHITE = "#FFFFFF"
RED = "#FF0000"
BLACK = "#000000"
NAVY = "#000080"
LIGHT BLUE = "#D9F2FF"
LIGHT GRAY = "#E1E1E1"
PINK = "#FFB6C1"
ORANGE = "#D58000"
GREEN = "#008000"

- **NOTE**: At this time, the HomeSeer colors for Error, Warning, and Updater log entries of Red, Orange, and Green respectively are automatic and are still set by HomeSeer.

Returns

None.

Example

```
Sub Main()

     hs.WriteLogEx "Error", "An error has occurred in my script!"
     hs.WriteLogEx "Hello", "I much prefer to see this in Navy Blue!", "#000080"

End Sub
```

See Also

WriteLogDetail

Purpose

This procedure writes an entry to the HomeSeer log with additional detailed information which can be used by plug-ins and on the log screen to highlight specific log events.

Parameters

Parameter: **mType**
Type: **String**
Description: This is the log entry "type", which is first to appear in the log. Usually this is used to indicate a severity or what the log entry pertains to, such as "Error" or "My Script".

Parameter: **Message**
Type: **String**
Description: This is the main log message.

Parameter: **Color**
Type: **String**
Description: This is the color code that you want associated with the log entry when you view it in the web browser. The color code must be in the #XXXXXX format, which is the # symbol followed by hexadecimal values for Red, Green, and Blue. Here are some examples of colors and their color values:

```
WHITE = "#FFFFFF"
RED = "#FF0000"
BLACK = "#000000"
NAVY = "#000080"
LIGHT BLUE = "#D9F2FF"
LIGHT GRAY  = "#E1E1E1"
PINK = "#FFB6C1"
ORANGE = "#D58000"
GREEN  = "#008000"
```

- **NOTE**: At this time, the HomeSeer colors for Error, Warning, and Updater log entries of Red, Orange, and Green respectively are automatic and are still set by HomeSeer.

Parameter: **Priority**
Type: **Integer**
Description: This is an indicator of the priority of the log entry, with the value 0 being unspecified, and the value 1 being the highest priority. Even if a process (e.g. Event) has logging turned off, priority 1 log entries are still written to the log.

Parameter: **mFrom**
Type: **String**
Description: This indicates the source of the message. For example: "Cool_Plugin, Main Procedure, Update Section"

Parameter: **ErrorCode**
Type: **Integer**
Description: This is an error code number which is meaningful only to the script or plug-in that generated this log entry.

Returns

Return value: None.

Example

```
hs.WriteLogDetail("Error", "Oh No, Mr. Bill!", COLOR_RED, 1, "SaturdayNight Plugin", 911)
```

See Also

ClearLog
GetLog
LogGet
NoLog
WriteLog
WriteLogEx

Web Pages

In This Section

GetPageFooter
GetPageHeader
WebValidateUser
WebStatsPageViews
WebServerSSLPort
WebServerPort
WebLoggedInUser
GetUsers
GetPlugLinks
RegisterHelpLink
RegisterLinkEx
UnRegisterHelpLinks

See Also

System Information
System Functions
INI File Editing
Plug-Ins
Logging
Callbacks
Launch
SendMessage
ReplaceVariables

GetPageFooter

Purpose

Returns the HomeSeer generated page footer for use in creating your own web pages.

Parameters

Optional Parameter: **NoEndTags**
Type: **Boolean**
Description: If set to TRUE, the html ending tags /BODY and /HTML will be omitted from the output. The default value if this parameter is not provided is FALSE.

Returns

The output is a string of HTML that comprises the HomeSeer generated page ending (footer) for web pages. The output consists of these elements:

- The navigation links, if the configuration is set to display them at the bottom of the web page, enclosed in a "navbottom" SPAN tag.
- The contents of the tail.htm file, enclosed in a "tailfile" SPAN tag.
- The </body> and </html> closing tags, unless the "NoEndTags" parameter is TRUE.

See Also

GetPageHeader
WebValidateUser
WebStatsPageViews
WebServerSSLPort
WebServerPort
WebLoggedInUser
GetUsers
GetPlugLinks
RegisterHelpLink
RegisterLinkEx
UnRegisterHelpLinks

GetPageHeader

Purpose

This procedure, useful when creating your own Active Server web Pages (ASPs) allows you to use HomeSeer's page header generating function with parameters that allow you to get all or part of the header information.

- As this procedure has options for returning the header with the HTML, HEAD, and BODY tags, you should make sure your use of this procedure does not generate duplicates of the above tags in your resulting page.

Parameters

Parameter: **title**
Type: **string**
Description: The title for the web page to be displayed at the top of the page before the logo bar.

Parameter: **extra_meta**
Type: **string**
Description: This parameter allows you to specify additional HTML to be included in the HEAD section of the page, and it should be formatted as a complete <meta ...> tag.

Parameter: **HSOnload**
Type: **string**
Description: This parameter is used by HomeSeer to specify Body_OnLoad procedures in the resulting page. They are prepended to the "Web Site" configuration item of the same name.

Parameter: **ExcludeNavLinks**
Type: **Boolean**
Description: If TRUE is passed to this parameter, the navigation links will not be included in the output.

Parameter: **NoHeader**
Type: **Boolean**
Description: If TRUE is passed to this parameter, the HTML tag and all contents of the HEAD section will be excluded from the output. The HEAD section includes the META tags, the page title, and the BODY tag (and thus Body_OnLoad is excluded).

Optional Parameter: **HeadContentOnly**
Type: **Boolean**
Description: If set to TRUE, only the contents of the HEAD html tag will be returned - use this if you are generating the other page elements yourself. See the "NoHeader" parameter for information on what is included in the HEAD tag section. The default value if this parameter is not specified is FALSE.

Optional Parameter: **BodyContentOnly**
Type: **Boolean**
Description: If set to TRUE, only the contents of the BODY html tag will be returned - this includes any Body_OnLoad specifications that are passed with the "HSOnload" parameter or the user's Body_OnLoad configuration value. This is useful when generating your own web pages but wish to maintain the user's Body_OnLoad options which may be used with other plug-ins in the system. The BODY tag is included in the output. The default value if this parameter is not specified is FALSE.

Optional Parameter: **BodyOnLoadOnly**
Type: **Boolean**
Description: If set to TRUE, only the contents of the "HSOnload" parameter and the user's Body_OnLoad configuration value. See "BodyContentOnly" for a usage scenario. The BODY html tag is not included in the output. The default value if this parameter is not specified is FALSE.

Returns

A string value containing the HTML content specified through the parameter choices.

A summary of a complete HomeSeer generated page header are as follows:

- HTML Tag
- HEAD Tag
- HomeSeer expiration and cache META tags, and any user specified HTML from the file "Web Site" configuration or META.HTM file.
- The TITLE tag and the title of the page.
- The BODY tag and any additional Body_OnLoad procedure specifications from the "Web Site" configuration.
- The contents of the HEAD.HTM file if it exists.
- The HomeSeer logo table area, enclosed in a "logotable" SPAN tag, and including these elements:
  - The page title portion of the logo table area, enclosed in a "pgtitle" SPAN tag.
  - The clock portion of the logo table area, enclosed in a "clock" SPAN tag, and an empty "userclock" SPAN tag for use in replacing the HomeSeer clock with a user generated version.
  - The sunrise portion of the logo table area, enclosed in a "lbsunrise" SPAN tag.
  - The sunset portion of the logo table area, enclosed in a "lbsunset" SPAN tag.
  - The logged on user portion of the logo table area enclosed in a "lbuser" SPAN tag.
- The navigation links, if specified to be included in the top of the page, enclosed in a "navtop" SPAN tag.

See Also

GetPageFooter
WebValidateUser
WebStatsPageViews
WebServerSSLPort
WebServerPort
WebLoggedInUser
GetUsers
GetPlugLinks
RegisterHelpLink
RegisterLinkEx
UnRegisterHelpLinks

WebValidateUser

Purpose

Returns TRUE if the given username/password pair is valid for the web server. Useful if you create your own login ASP web page.

Parameters

Parameter: **username**
Type: **string**
Description: Name of the user to validate.

Parameter: **password**
Type: **string**
Description: Password of the user to validate.

Returns

Return value: **user authorization**
Type: **boolean**

See Also

WebStatsPageViews

Purpose

This is a read/write property. It will return the number of times your web site has displayed a complete page. To reset the statistics, set this property to 0.

Parameters

Parameter: **=value**
Type: **string**
Note: Set to 0 when clearing the stats.

Returns

Return value: **page statistics**
Type: **integer**
Description: The number of page views from the HomeSeer web site as an integer.

Example

```
' get the page view stats and set to a virtual device for display

sub main()

      dim s

      s = hs.WEBStatsPageViews
      hs.SetDeviceString "z1","Page Views: "&cstr(s)

end sub


' reset the stats

sub main()

      hs.WEBStatsPageViews = 0

end sub
```

See Also

GetPageFooter
GetPageHeader
WebValidateUser
WebServerSSLPort
WebServerPort
WebLoggedInUser
GetUsers
GetPlugLinks
RegisterHelpLink
RegisterLinkEx
UnRegisterHelpLinks

WebServerSSLPort

Purpose

Returns the port number of the HomeSeer SSL (Secure Socket Layer) Web Server if enabled.

Parameters

None.

Returns

Return value: **port**
Type: **integer**
Description: The port number if the SSL server is enabled, otherwise 0.

See Also

GetPageFooter
GetPageHeader
WebValidateUser
WebStatsPageViews
WebServerPort
WebLoggedInUser
GetUsers
GetPlugLinks
RegisterHelpLink
RegisterLinkEx
UnRegisterHelpLinks

WebServerPort

Purpose

Provides the port number of the HomeSeer web server.

Parameters

None.

Returns

Return value: **port**
Type: **integer**
Description: The port number.

See Also

GetPageFooter
GetPageHeader
WebValidateUser
WebStatsPageViews
WebServerSSLPort
WebLoggedInUser
GetUsers
GetPlugLinks
RegisterHelpLink
RegisterLinkEx
UnRegisterHelpLinks

WebLoggedInUser

Purpose

Returns the user ID of the person who is logged into the web server. This is useful for scripts that you may not want to run if a guest is logged in.

Parameters

None.

Returns

Return value: **current user**
Type: **string**

See Also

GetPageFooter
GetPageHeader
WebValidateUser
WebStatsPageViews
WebServerSSLPort
WebServerPort
GetUsers
GetPlugLinks
RegisterHelpLink
RegisterLinkEx
UnRegisterHelpLinks

GetUsers

Purpose

Returns a list of web users and their access rights (but not passwords) from the system. The list returned is in the form: `username|rights,username2|rights2`, etc.

User rights are values that are sometimes OR'd together and are as follows:

USER_GUEST = 1

USER_ADMIN = 2
USER_LOCAL = 4
USER_NORMAL = 8

A user is either GUEST, NORMAL, or ADMIN, but any of the user IDs with that access level can also be the LOCAL user, which is the username used when the system is accessed via the local network. For example, a user with rights that are equal to 10 (decimal) is the local user, and that user has admin rights.

- This function returns an empty string if the command has not been permitted in the web server configuration options, Security setting.

Parameters

None.

Returns

Return value: **user list**
Type: **string**

Example

This script will produce a list of the users and their rights to the system log.

```
sub main()

    Dim sAllUsers
    Dim sUserPairs
    Dim i
    Dim sTemp
    Dim sUser
    Dim iRights
    Dim sRights
    Dim bNoRights
    CONST USER_GUEST = 1
    CONST USER_ADMIN = 2
    CONST USER_LOCAL = 4

    sAllUsers = hs.GetUsers

    sUserPairs = Split(sAllUsers,",")
    ' Now sUserPairs is an array of username,rights pairs.

    for i = 0 to UBound(sUserPairs)

        sTemp = sUserPairs(i)
        sUser = left(sTemp,instr(sTemp,"|") - 1)
        iRights = cint(trim(mid(sTemp,instr(sTemp,"|")+1)))
        sRights = ""
        bNoRights = False
        if (iRights and USER_GUEST) = USER_GUEST then
           sRights = sRights & "Guest"
        end if
        if (iRights and USER_ADMIN) = USER_ADMIN then
           sRights = sRights & "Admin"
        end if
        if len(sRights) = 0 then
           bNoRights = True
        end if
        if (iRights and USER_LOCAL) = USER_LOCAL then
           if len(sRights) > 0 then
               sRights = sRights & " and this is the Local Login ID"
           else
               sRights = "Local Login ID"
           end if
        end if

        if bNoRights then
           hs.writelog "User Info","Name is: " & sUser & " and has no user rights. " & sRig
        else
           hs.writelog "User Info","Name is: " & sUser & " and the rights are: " & sRights
        end if

    next

end sub
```

Example output from this script:

4/1/2004 12:00:00 AM~!~Event Trigger~!~Trigger from menu (GetUsers Test)
4/1/2004 12:00:00 AM~!~User Info~!~Name is: guest and has no user rights.
4/1/2004 12:00:00 AM~!~User Info~!~Name is: Mary and has no user rights. Local Login ID
4/1/2004 12:00:00 AM~!~User Info~!~Name is: Charlie and the rights are: Admin

See Also

GetPageFooter
GetPageHeader
WebValidateUser
WebStatsPageViews
WebServerSSLPort
WebServerPort
WebLoggedInUser
GetPlugLinks
RegisterHelpLink
RegisterLinkEx
UnRegisterHelpLinks

GetPlugLinks

Purpose

Returns a list of plug-in web page titles and link locations, separated by chr(1) and chr(2) characters.

Parameters

None.

Returns

Return value: **plug-in link pages information**
Type: **string**

Example

```
sub main()

    dim pname
    dim plink
    dim sTemp

    sTemp = hs.GetPlugLinks
    plink = split(sTemp,chr(2))
    for x = 0 to ubound(plink)
        pname = split(plink(x),chr(1))
        hs.writelog "GetPlugLinks","Link titled " & pname(0) & " is at " & pname(1)
    next

end sub
```

See Also

GetPageFooter
GetPageHeader
WebValidateUser
WebStatsPageViews
WebServerSSLPort
WebServerPort
WebLoggedInUser
GetUsers
RegisterHelpLink
RegisterLinkEx
UnRegisterHelpLinks

RegisterHelpLink

```
Sub RegisterHelpLink(ByVal cbo As WebPageDesc)
```

Purpose

This call registers a help link resource with HomeSeer so that it will appear on the help (/help) page of HomeSeer. This function can be used by both plug-ins and scripts.

Parameters

Parameter: **CBO**
Type: **Call-Back Object**
Description: This class object contains several parameters used by web page link definitions. See WebPageDesc for more information on the properties within this object.

Returns

None.

Example

```
Sub Main(ByVal Parms As Object)

    Dim cbo As New WebPageDesc
    cbo.plugInName = "UserScript1"
    cbo.plugInInstance = "Help Page Link"
    cbo.link = "MyScript/Help/MyHelpFile.pdf"
    cbo.linktext = "Utility Script System Help"
    cbo.page_title = "Utility System Help Page"

    hs.RegisterLinkEx(cbo)

End Sub
```

See Also

GetPageFooter
GetPageHeader
WebValidateUser
WebStatsPageViews
WebServerSSLPort
WebServerPort
WebLoggedInUser
GetUsers
GetPlugLinks
RegisterLinkEx
UnRegisterHelpLinks

WebPageDesc Object

This class object is used by several functions which register web pages or links to web pages in the HomeSeer web server or web page menus.

```
Public Class WebPageDesc
    Public plugInName As String = ""  ' When used by a script call to register a non-plugin link,
    '                                   this is a name to be associated with the link so that
    '                                  links can be grouped and for the removal of links done
    '                                   via other calls.
    Public plugInInstance As String = ""     ' When used by a script call to register a non-plugin link,
    '                                   this is a unique string that can be used to unregister
    '                                  an individual link from a group of links registered under
    '                                   the same plugInName.
```

```
    Public link As String = ""    ' The link to be registered. For example, "MyAboutPage",
    '                                   once registered, could be accessed using:
    '                                   http://(HomeSeer:Port)/MyAboutPage
    Public linktext As String = ""          ' The text to appear in the HomeSeer menu system for the link.
    Public page_title As String = ""        ' The title to be displayed for the web page.
    Public order As Integer               ' Used by RegisterHelpLink only to determine the display order of help links.
End Class
```

See Also

RegisterLinkEx

```
Sub RegisterLinkEx(ByVal cbo As WebPageDesc)
```

Purpose

This call registers a link resource with HomeSeer so that it will appear on the menu bar of HomeSeer. This function can be used by both plug-ins and scripts.

Parameters

Parameter: **CBO**
Type: **Call-Back Object**
Description: This class object contains several parameters used by web page link definitions. See WebPageDesc for more information on the properties within this object.

Returns

None.

Example

```
Sub Main(ByVal Parms As Object)

    Dim cbo As New WebPageDesc
    cbo.plugInName = "UserScript1"
    cbo.plugInInstance = "Configuration Page Link"
    cbo.link = "UtilityConfig"
    cbo.linktext = "Utility Script System Configuration"
    cbo.page_title = "Utility System Configuration Page"

    hs.RegisterLinkEx(cbo)

End Sub
```

See Also

GetPageFooter
GetPageHeader
WebValidateUser
WebStatsPageViews
WebServerSSLPort
WebServerPort
WebLoggedInUser
GetUsers
GetPlugLinks
RegisterHelpLink
UnRegisterHelpLinks

WebPageDesc Object

This class object is used by several functions which register web pages or links to web pages in the HomeSeer web server or web page menus.

```
Public Class WebPageDesc
    Public plugInName As String = ""  ' When used by a script call to register a non-plugin link,
    '                                   this is a name to be associated with the link so that
    '                                  links can be grouped and for the removal of links done
    '                                   via other calls.
    Public plugInInstance As String = ""     ' When used by a script call to register a non-plugin link,
    '                                   this is a unique string that can be used to unregister
    '                                  an individual link from a group of links registered under
    '                                   the same plugInName.
    Public link As String = ""   ' The link to be registered. For example, "MyAboutPage",
    '                                   once registered, could be accessed using:
    '                                   http://(HomeSeer:Port)/MyAboutPage
    Public linktext As String = ""           ' The text to appear in the HomeSeer menu system for the link.
    Public page_title As String = ""         ' The title to be displayed for the web page.
    Public order As Integer                ' Used by RegisterHelpLink only to determine the display order of help links.
End Class
```

See Also

UnRegisterHelpLinks

Purpose

This call removes all of the registered help resource links for the plug-in or script/ASPX registered with the provided name. See RegisterHelpLink for more information on registering a help resource.

- This procedure is only valid in HomeSeer HS2 versions after 2.2.0.0.
- Help resources that exist on the hard drive such as a static html document do not need to be explicitly unregistered. However, when a help resource is provided by a plug-in or when the help resource requires the use of a plug-in, then this procedure should be used to unregister the resource when the plug-in shuts down (e.g. ShutdownIO) so the user does not have a link displayed that will not work properly.

Parameters

Parameter: **plug-in name**
Type: **String**
Description: This is the plug-in name or some sort of unique identifier for a script or ASPX based system. This identifier is used to group multiple links from the same plug-in or script/ASPX together, and it is displayed as a heading on the help page. It is not required that a plug-in use its IFACE_NAME value, but it is necessary to use the same text here as when you used RegisterHelpLink to register the link in the first place.

Returns

None.

Example

To unregister all help resources for the Acme_Widgets plug-in, which were registered using a plug-in name of "Acme Application"

```
hs.UnRegisterHelpLinks("Acme Application")
```

See Also

GetPageFooter
GetPageHeader
WebValidateUser
WebStatsPageViews
WebServerSSLPort
WebServerPort
WebLoggedInUser
GetUsers
GetPlugLinks
RegisterHelpLink
RegisterLinkEx

Callbacks

In This Section

RegisterStatusChangeCB
UnRegisterStatusChangeCB

See Also

System Information
System Functions
INI File Editing
Plug-Ins
Logging
Web Pages
Launch
SendMessage
ReplaceVariables

RegisterStatusChangeCB

Purpose

HomeSeer has the ability to trigger events based on a device changing. It may be useful to run a script when a device changes. The `RegisterStatusChangeCB` function can be used to register your script with HomeSeer. When a device changes, your script will be called. The script is passed the code of the device that changed, the address of the device that changed, as well as the value the device changed to and the reference ID of the device.

To remove the callback script, call hs.UnRegisterStatusChangeCB. There are no parameters with this call.

Remarks

When a device changes status, the given script is called as follows:

```
script_name( parm )
```

The `parms` parameter is an array of parameters. The following parameters are available:

parm(0) = Code part of the Address of the device that changed status
parm(1) = Full Address of the device that changed status (including Code if present)
parm(2) = The New value of the device that changed status.
parm(3) = The Old value of the device that changed status.
parm(4) = The device reference ID number. Can be used with GetDeviceByRef to find the DeviceClass of the specific device.

Note that since a function can be called in the callback script, the registration and actual callback can all reside in the same script file.

Parameters

Parameter: **Script**
Type: **String**
Description: This is the file name of the script to run. Do not include the path in the script name; the script is assumed to be in the scripts directory (`C:\Program Files\HomeSeer HS3\Scripts` by default).

Parameter: **Function**
Type: **String**
Description: This is the function in the script to run, such as `Main`.

Returns

Return: **Result**
Type: **Boolean**
Description: If True, the registration of the script succeeded.

Example

```
' register a callback script

Sub Main(ByVal Parms As Object)

   hs.RegisterStatusChangeCB("Stat_Change.vb", "StatusChangeCB")

End Sub


' the actual status change script that is called when a device changes status given the above register
call

Sub StatusChangeCB(ByVal Parm As Object())

   If Parm Is Nothing Then Exit Sub
   If Parm.Length < 5 Then Exit Sub

   Dim Code As String = ""
   Dim Address As String = ""
   Dim OldVal As Double
   Dim NewVal As Double
   Dim Ref As Integer

   Try
      Code = Parm(0).ToString
      Address = Parm(1).ToString
      NewVal = Parm(2)
      OldVal = Parm(3)
      Ref = Parm(4)
   Catch ex As Exception
      hs.WriteLog("Stat_Change.VB", "Error, Exception parsing Parm: " & ex.Message)
      Exit Sub
   End Try

   hs.WriteLog("Change", hs.DeviceName(Ref) & " changed from " & OldVal.ToString & " to " & _
                         NewVal.ToString & " (" & Address & ")")

End Sub
```

See Also

UnRegisterStatusChangeCB

UnRegisterStatusChangeCB

Purpose

This function removes a script associated with a status change as set with RegisterStatusChangeCB.

Parameters

Parameter: **Script**
Type: **String**
Description: This is the name of the script file provided when RegisterStatusChangeCB was called.

Returns

None.

See Also

RegisterStatusChangeCB

Launch

```
Function Launch(ByVal Name As String, _
                ByVal Params As String, _
                ByVal Directory As String, _
                ByVal LaunchPri As Integer) As Integer
```

Purpose

Launches a given application. The function will return before the application finishes launching.

Parameters

Parameter: **Name**
Type: **String**
Description: This is the name of the EXE file to launch. It can be a simple application name (the path to the application would have to be in your system path) or it can be a full path name to the file. Application files can also be launched and the application that owns the file will be executed.

Parameter: **Params**
Type: **String**
Description: Any parameters or command line switches that are to be passed to the application.

Parameter: **Directory**
Type: **String**
Description: The working directory the application is launched from. Leave an empty string for most applications.

Parameter: **LaunchPri**
Type: **Integer**
Description: The running priority of the process. Use -1 for Below Normal, 0 for Normal, 1 for Above Normal.

Returns

Return value: **Process Instance**
Type: **Integer**
Description: The instance number of the process (not very useful).

See Also

System Information
System Functions
INI File Editing
Plug-Ins
Logging
Web Pages
Callbacks
SendMessage
ReplaceVariables

Home > Scripting > Applications and Plugins > SendMessage

SendMessage

Purpose

This command transmits a text message to speaker clients and controls how it is to be displayed.

- This command is only valid with the Professional edition of HomeSeer.

Parameters

Parameter: **message**
Type: **string**
Description: The text of the message you wish to send.

Parameter: **host**
Type: **string**
Description: The host name, comma separated hosts list, or comma separated list of host:instance names that you wish to send the message to.

Parameter: **showballoon**
Type: **boolean**
Description: If set to TRUE, the text will be displayed in a balloon popup window in the system tray.

Returns

Return value: **error**
Type: **integer (.NET Enum, Short)**
Description: 1 = No Error, 2 = There are no speaker clients to send to, 3 = An error occurred during the sending.

Example

```
Sub Main()

      Dim i

      i = hs.SendMessage("Hello, World.", "*:Default", True)
      if i > 1 then
        hs.WriteLog "Error","SendMessage failed with code " & CStr(i)
      end if

End Sub
```

See Also

System Information
System Functions
INI File Editing
Plug-Ins
Logging
Web Pages
Callbacks
Launch
ReplaceVariables

ReplaceVariables

Purpose

Does string variable replacement in a script. The variables that can be replaced are the same as those listed here.

Parameters

Parameter: **InputString**
Type: **string**
Description: The string with special variables to be modified/replaced with values.

Returns

Return value: **OutputString**
Type: **string**
Description: The string with the new values in place of the indicated replacement variables.

See Also

System Information
System Functions
INI File Editing
Plug-Ins
Logging
Web Pages
Callbacks
Launch
SendMessage

Using Replacement Variables

Replacement variables are a series of special characters that you can use in text being spoken or in the subject or body of an email. When HomeSeer encounters one of these variables, it substitutes the information indicated by the variable in place of the variable.

Example

hs.Speak "The time is $$time"

Results in (at 11AM): "The time is <say-it type="time"> 11:00 AM </say-it>"

Or to do the same without the prosody/pronunciation tag:
hs.Speak "The time is $time"

Results in (at 11AM): "The time is 11:00 AM"

HomeSeer Replacement Variables

(Replacement Variables are Case Insensitive)

$date	Replacement is the current date in long format, e.g.: April 1, 2006
$time	Replacement is the current time in 12 hour format, e.g. 2:00 PM
$$date	Replacement is the same as $date, but it is wrapped with the SAPI context tag for date so the text to speech engine knows it is a date spoken. Use $$date when the output is going to be spoken.

$$time	Replacement is the same as $time, but it is wrapped with the SAPI context tag for time so the text to speech engine knows it is a time b spoken. Use $$time when the output is going to be spoken.
$from	Replacement is the email address of the last email received.
$$DVA:(address)**:**	Replacement is the VALUE of the device indicated by (address). For example, if the device at address 003E2CDD-R40 has a value of 100, then using $$DVA:003E2CDD-R40: in the text will result in 100 after the substitution. • **Note**: This replacement variable is terminated with a second colon (:) at the end of the address.
$$DVC:(code)**:**	Replacement is the VALUE of the device indicated by (code). For example, if the device at code R40 has a value of 100, then using $$DVC:R40: in the text will result in 100 after the substitution. • **Note**: This replacement variable is terminated with a second colon (:) at the end of the code.
$$DVR:(dvRef)**:**	Replacement is the VALUE of the device indicated by (reference). For example, if the device at device reference 5236 has a value of 1 then using $$DVR:5236: in the text will result in 100 after the substitution. • **Note**: This replacement variable is terminated with a second colon (:) at the end of the reference.
$$DSA:(address)**:**	Replacement is the STATUS of the device indicated by (address). For example, if the device at address 003E2CDD-S39 has a status of "Disarmed", then using $$DSA:003E2CDD-S39: in the text will result in "Disarmed" after the substitution. • **Note**: HTML used in the status may result in problems when the replaced text is spoken. • **Note**: This replacement variable is terminated with a second colon (:) at the end of the address.
$$DSC:(code)**:**	Replacement is the STATUS of the device indicated by (code). For example, if the device at code S39 has a status of "Disarmed", then $$DSC:S39: in the text will result in "Disarmed" after the substitution. • **Note**: HTML used in the status may result in problems when the replaced text is spoken. • **Note**: This replacement variable is terminated with a second colon (:) at the end of the code.
$$DSR:(dvRef)**:**	Replacement is the STATUS of the device indicated by (reference). For example, if the device at reference 4849 has a status of "Disarr then using $$DSR:4849: in the text will result in "Disarmed" after the substitution. • **Note**: HTML used in the status may result in problems when the replaced text is spoken. • **Note**: This replacement variable is terminated with a second colon (:) at the end of the reference.
$$LCI:(line)	Replacement is the **L**ast **C**aller **I**nformation for the indicated (line) number. See LastCallerInfo for information on what is substituted.
$$CIN:(line)	Replacement is the last **C**aller **I**dentification **N**ame received for the indicated (line) number.
$$CI#:(line)	Replacement is the last **C**aller **I**dentification Number for the indicated (line) number.
$$LVM	Replacement is the **L**ast **V**oice**M**ail information for the last voicemail left by a caller.

See Also

Computer

In This Section

Serial Port Communication
Network Information
GetOSVersion
RecurseFiles
RecurseFilesEx
RestartSystem
UnZip
Zip
Keys

See Also

About Scripts
Applications and Plugins
Devices
Email
Events
Internet
Phone
Scripts
Speech Recognition
Strings, Global Variables, and Encryption
Time and Calendar
Text-To-Speech and Media

Home > Scripting > Computer > Serial Port Communication

Serial Port Communication

See Also

Network Information
GetOSVersion
RecurseFiles
RecurseFilesEx
RestartSystem
UnZip
Zip
Keys

Home > Scripting > Computer > Serial Port Communication > OpenComPort

OpenComPort

Purpose

This function opens a communication port. If the port is already open by another application, an error occurs. Once a port is opened, it remains open until the CloseComPort function is called. The port is not closed when the script terminates. However, the port will close when the application terminates.

- In previous versions of HomeSeer, the `OpenComPort` function was limited to COM ports 1 to 8 and OpenComPortEx was for ports above 8. Beginning with HomeSeer Version 2.0, there is no limit on the number of COM ports, so these functions can be used interchangeably. The "Resource" parameter in the `OpenComPortEX` function is not used in HomeSeer 2.0, but the parameter is included in order to support scripts created with older versions of HomeSeer.
- If you need to control the hardware handshaking signals `Request To Send (RTS)` or `Data Terminal Ready (DTR)`, please refer to the SetComPortRTSDTR command.

Parameters

Parameter: **port**
Type: **integer**
Description: This is the port number to open. An error is returned if the port is already open or is not installed on the system. To use port numbers above 8, see the resource description below.

Parameter: **config**
Type: **string**
Description: Port Configuration (see below).

Parameter: **mode**
Type: **integer**
Description: Operating Mode (see below).

Parameter: **cb_script**
Type: **string**
Description: Port Data Handling Script (see below).

Parameter: **cb_func**
Type: **string**
Description: Function in Port Handling Script (see below).

Parameter: **term** (optional)
Type: **string**
Descripton: Termination String (see below).

Returns

The function returns an empty string if it was successful, otherwise it returns a text string describing the error.

Port Configuration

The `config` parameter is composed of four settings and has the format `BBBB,P,D,S`.

`BBBB` is the baud rate, `P` is the parity, `D` is the number of data bits, and `S` is the number of stop bits. For example, to set the port to 9600 baud, no parity, 8 bit and no stop bits, the `config` string would be `9600,N,8,1`.

The valid baud rates are listed below.

110	2400	19200	57600
300	4800	28000	115200
600	9600	38400	128000
1200	14400	56000	256000

The parity values are:

E = Even
M = Mark
N = None
O = Odd
S = Space

The data bit values are:

5
6
7
8

The stop bit values are:

1
1.5
2

Operating Mode

This parameter affects the way data is received on the COM port. Two modes are available:

0 = raw mode

In this mode, each character that is received on the COM port causes the specified script and function to be called. It is up to the called function to call GetComPortData to actually get the characters.

1 = strings mode

This mode buffers up characters until a terminator is received. At this point the specified script and function are called with the data. This mode makes it easy to deal with devices that send text data terminated with known characters. To specify the terminator characters, see the term parameter description below. If you do not specify a terminator, the default terminator of carriage return and line-feed (CrLf) are used.

Port Data Handling Script

This parameter is the name of the script that will be called when COM port data arrives. The script will be called with a single parameter, which is the received text string. If you do not wish to be called back when data is received, leave this parameter as an empty string. You can still use the

GetComPortData function to poll for data yourself. The following example shows what your called script should look like.

```
sub callback(data)
' handle the data
end sub
```

Function in Port Data Handling Script

This is the function that will be called in the specified script. If your script was defined as above, the `cb_func` parameter would be set to `callback`. If this parameter is omitted, the `main` function will be called by default.

Termination String

This is the terminator string for mode 1 operation. Characters will be received into the COM port buffer until this termination string is found in the buffer. If this parameter is not provided, then the default value is the character pair of carriage return and line-feed (CrLf) .

See Also

OpenComPortEx
SetComPortRTSDTR
SendToComPort
GetComPortCount
GetComPortData
CloseComPort

OpenComPortEx

Purpose

This function opens a communication port. If the port is already open by another application, an error occurs. Once a port is opened, it remains open until the CloseComPort function is called. The port is not closed when the script terminates. However, the port will close when the application terminates.

- In previous versions of HomeSeer, the OpenComPort function was limited to COM ports 1 to 8 and `OpenComPortEx` was for ports above 8. Beginning with HomeSeer Version 2.0, there is no limit on the number of COM ports, so these functions can be used interchangeably. The "Resource" parameter is not used in HomeSeer 2.0, but the parameter is included in order to support scripts created with older versions of HomeSeer.
- If you need to control the hardware handshaking signals `Request To Send (RTS)` or `Data Terminal Ready (DTR)`, please refer to the SetComPortRTSDTR command.

Parameters

Parameter: **port**
Type: **integer**
Description: This is the port number to open. An error is returned if the port is already open or is not installed on the system. To use port numbers above 8, see the resource description below.

Parameter: **config**
Type: **string**
Description: Port Configuration (see below).

Parameter: **mode**
Type: **integer**
Description: Operating Mode (see below).

Parameter: **cb_script**
Type: **string**
Description: Port Data Handling Script (see below).

Parameter: **cb_func**
Type: **string**
Description: Function in Port Handling Script (see below).

Parameter: **term** (optional)
Type: **string**
Descripton: Termination String (see below).

Parameter: **resource** (optional)
Type: **integer**
Description: Resource Number (see below). This parameter is not used in HomeSeer 2.0 but is included for backward-compatibility with scripts created

in older versions of HomeSeer.

Returns

The function returns an empty string if it was successful, otherwise it returns a text string describing the error.

Port Configuration

The `config` parameter is composed of four settings and has the format `BBBB,P,D,S`.

`BBBB` is the baud rate, `P` is the parity, `D` is the number of data bits, and `S` is the number of stop bits. For example, to set the port to 9600 baud, no parity, 8 bit and no stop bits, the `config` string would be `9600,N,8,1`.

The valid baud rates are listed below.

110	2400	19200	57600
300	4800	28000	115200
600	9600	38400	128000
1200	14400	56000	256000

The parity values are:

E = Even
M = Mark
N = None
O = Odd
S = Space

The data bit values are:

5
6
7
8

The stop bit values are:

1
1.5
2

Operating Mode

This parameter affects the way data is received on the COM port. Two modes are available:

0 = raw mode

In this mode, each character that is received on the COM port causes the specified script and function to be called. It is up to the called function to call GetComPortData to actually get the characters.

1 = strings mode

This mode buffers up characters until a terminator is received. At this point the specified script and function are called with the data. This mode makes it easy to deal with devices that send text data terminated with known characters. To specify the terminator characters, see the term parameter description below. If you do not specify a terminator, the default terminator of carriage return and line-feed (CrLf) are used.

Port Data Handling Script

This parameter is the name of the script that will be called when COM port data arrives. The script will be called with a single parameter, which is the received text string. If you do not wish to be called back when data is received, leave this parameter as an empty string. You can still use the GetComPortData function to poll for data yourself. The following example shows what your called script should look like.

```
sub callback(data)
' handle the data
end sub
```

Function in Port Data Handling Script

This is the function that will be called in the specified script. If your script was defined as above, the `cb_func` parameter would be set to `callback`. If this parameter is omitted, the `main` function will be called by default.

Termination String

This is the terminator string for mode 1 operation. Characters will be received into the COM port buffer until this termination string is found in the buffer. If this parameter is not provided, then the default value is the character pair of carriage return and line-feed (CrLf).

Resource Number

This parameter is no longer necessary in HomeSeer 2.0 but is included for backward-compatibility with scripts created in older versions of HomeSeer.

This is a resource number to allocate `OpenComPortEx` resources so that `OpenComPortEx` can be used with COM ports above 8. There are 8 resources available between `OpenComPort` and `OpenComPortEx`. When you wish to use COM ports above 8 you can specify the higher COM port number for the port parameter, but then you must specify a resource number with this parameter. HomeSeer does NOT keep track of used resource numbers. If COM3 is opened with `OpenComPort`, which means resource 3 was assigned to it, you must remember not to use resource 3 with `OpenComPortEx`.

See Also

OpenComPort
SetComPortRTSDTR
SendToComPort
GetComPortCount
GetComPortData
CloseComPort

SetComPortRTSDTR

Purpose

Sets the levels of the RTS and DTR signals on the given COM port.

Parameters

Parameter: **port**
Type: **integer**
Description: This is the COM port to access or the resource number of the port to access if OpenComPortEx was used to open it.

Parameter: **rts_val**
Type: **boolean**
Description: Set to TRUE to raise the RTS line or set to FALSE to lower the line.

Parameter: **dtr_val**
Type: **boolean**
Description: Set to TRUE to raise the DTR line or set to FALSE to lower the line.

Returns

None.

See Also

OpenComPort
OpenComPortEx
SendToComPort
GetComPortCount
GetComPortData
CloseComPort

SendToComPort

Purpose

Send a string of characters out a communications port. The port must have been previously opened with the OpenComPort or OpenComPortEx call.

- Some devices that you are communicating with require a special character to terminate the string of characters you are sending to it. For example, a modem needs a carriage-return (CR) at the end of the string you send to it before it will be recognized. Some devices may require a carriage-return and a line-feed character, others perhaps something entirely different. Please be aware of the requirements of the device you are communicating with. If you require hardware handshaking on the communications port, please see the SetComPortRTSDTR command.

Parameters

Parameter: **port**
Type: **integer**
Description: This is the COM port to send the data on or the resource number of the port to send data on if OpenComPortEx was used to open it.

Parameter: **data**
Type: **string**
Description: This is the actual data to send out the COM port.

Returns

Return value: None

See Also

OpenComPort
OpenComPortEx
SetComPortRTSDTR
GetComPortCount
GetComPortData
CloseComPort

GetComPortCount

Purpose

Returns the number of received characters available on a communications port. This function can be used to poll the COM port for data. The best way to receive characters on a COM port is to use the callback function that is set with `OpenComPort` or `OpenComPortEx`.

Parameters

Parameter: **port**
Type: **integer**
Description: The port number of the port to check or the resource number of the port to be checked if `OpenComPortEx` was used to open it.

Returns

Return value: **number**
Type: **integer**
Description: The number of characters available at the COM port.

See Also

OpenComPort
OpenComPortEx
SetComPortRTSDTR
SendToComPort
GetComPortData
CloseComPort

GetComPortData

Purpose

Returns the data available at a COM port. The data is a variant and could be a text string or an array of bytes, depending on the type of data

available. This function is not used if the COM port is opened in mode 1. If the port is opened as mode 0, this function should be used in your callback function to get the data.

Parameters

Parameter: **port**
Type: **integer**
Description: The COM port to read or the resource number of the port to be read if OpenComPortEx was used to open it.

Returns

Return value: **data**
Type: **variant**
Description: The data available is a string of characters.

See Also

OpenComPort
OpenComPortEx
SetComPortRTSDTR
SendToComPort
GetComPortCount
CloseComPort

CloseComPort

Purpose

Closes a communications port previously opened with OpenComPort, or the communications port associated with a resource that was opened with the OpenComPortEx command.

Parameters

Parameter: **port**
Type: **integer**
Description: The number of the port to be closed or the resource number of the port to be closed if OpenComPortEx was used to open it.

Returns

None.

See Also

OpenComPort
OpenComPortEx
SetComPortRTSDTR
SendToComPort
GetComPortCount
GetComPortData

Network Information

See Also

Serial Port Communication
GetOSVersion
RecurseFiles
RecurseFilesEx
RestartSystem
UnZip
Zip
Keys

GetIPAddress

Purpose

Returns the IP address of your computer as a string like `192.168.1.1`. Note that if your computer has multiple network interfaces, this will return the IP address of each interface separated by a "space" character: `192.168.1.1 192.168.1.2`.

If you wish to get the IP address and hostname of the machine, please see LANIP.

Parameters

None.

Returns

Return value: **IP address**
Type: **string**

Example

```
sub main()

     dim ipaddress

     ipaddress = hs.GetIPAddress
     hs.WriteLog "Info","The IP Address is " & ipaddress

end sub
```

see Also

GetLastRemoteIP
LANIP
WANIP

See Also

GetLastRemoteIP
LANIP
Ping
WANIP

GetLastRemoteIP

Purpose

Returns the IP address of the last client computer to access the HomeSeer web server, as a string like `192.168.1.1`.

Parameters

None.

Returns

Return value: **IP address**
Type: **string**

Example

```
sub main()

     dim ipaddress

     ipaddress = hs.GetLastRemoteIP
     hs.WriteLog "Info","The IP Address to last access the system is " & ipaddress

end sub
```

see Also

LANIP
WANIP
GetIPAddress

See Also

GetIPAddress
LANIP
Ping
WANIP

Home > Scripting > Computer > Network Information > LANIP

LANIP

Purpose

Provides the IP address and hostname of the HomeSeer computer's primary network interface, as seen from the local (in house) network. If you want the IP address only, please see GetIPAddress.

Parameters

None.

Returns

Return value: **IP**
Type: **string**
Description: The IP address and hostname is returned in the format: xxx.xxx.xxx.xxx (hostname)

See Also

GetIPAddress
GetLastRemoteIP
WANIP

See Also

GetIPAddress
GetLastRemoteIP
Ping
WANIP

Ping

Purpose

Indicates of a host is available.

Parameters

Parameter: **host name**
Type: **string**
Description: Name or IP address of the host to ping.

Returns

Return value: **host status**
Type: **integer**
Description: Returns 0 if host is alive and 26118 if host is not available.

See Also

GetIPAddress
GetLastRemoteIP
LANIP
WANIP

WANIP

Purpose

Provides the IP address and hostname of the HomeSeer computer's primary network interface as seen from the Internet.

Parameters

None.

Returns

Return value: **IP**
Type: **string**
Description: The IP address and hostname is returned in the format: xxx.xxx.xxx.xxx (hostname)

See Also

GetIPAddress
GetLastRemoteIP
LANIP

See Also

GetIPAddress
GetLastRemoteIP
LANIP
Ping

GetOSVersion

Purpose

Returns the version of the operating system running HomeSeer.

Parameters

None.

Returns

Return value: **OS Version**
Type: **string**
Example: **5.1.0.2600**

Example

```
sub main()

     hs.WriteLog "Info","The Operating System version is " & hs.GetOSVersion

end sub
```

See Also

Serial Port Communication
Network Information
RecurseFiles
RecurseFilesEx
RestartSystem
UnZip
Zip
Keys

RecurseFiles

Purpose

This command returns a comma separated string of files that are in the starting directory and all sub-directories within it.

Parameters

Parameter: **Starting Directory**
Type: **string**
Description: The full path to the starting directory to be recursed.

Returns

Parameter: **file list**
Type: **string**
Description: The list of files in the starting directory and sub-directories, separated by a comma.

See Also

Serial Port Communication
Network Information
GetOSVersion
RecurseFilesEx
RestartSystem
UnZip
Zip
Keys

RecurseFilesEx

Function RecurseFilesEx(ByVal SourceDir As String) As String()

Purpose

This command returns an array of strings that are in the starting directory and all sub-directories within it.

Parameters

Parameter: **Starting Directory**
Type: **string**
Description: The full path to the starting directory to be recursed.

Returns

Parameter: **file list**
Type: **string array**
Description: The list of files in the starting directory and sub-directories, one entry per array element.

See Also

Serial Port Communication
Network Information
GetOSVersion
RecurseFiles
RestartSystem
UnZip
Zip
Keys

RestartSystem

Purpose

This command will shut down HomeSeer and restart your computer system. **Use with caution!**

The shut down command for HomeSeer is coming from the computer system. Thus, it is possible for the system to restart before HomeSeer has completed its shut down processing.

Parameters

None.

Returns

None.

See Also

Serial Port Communication
Network Information
GetOSVersion
RecurseFiles
RecurseFilesEx
UnZip
Zip
Keys

UnZip

Purpose

This command will unzip a Zip archive file to the destination you provide.

Parameters

Parameter: **filename**
Type: **string**
Description: Path and name of the source Zip archive file to be unzipped.

Parameter: **destination** (optional)
Type: **string**
Description: Path to the destination starting directory for the files in the zip archive. If this parameter is not provided, the files in the Zip archive will be unzipped to the same directory as the source zip file.

Parameter: **IgnoreZipDirs** (optional)
Type: **boolean**
Description: If True, the zip directories within the Zip archive will be ignored and all of the files will be unzipped into the destination directory only. (Note: Two files of the same name in different Zip archive directories will result in only one of them existing at the end of the UnZip operation if this parameter is set to True.) (Default is False)

Parameter: **OverWrite** (optional)
Type: **boolean**
Description: If set to True, existing destination files will be overwritten. (Default is False)

Parameter: **password** (optional)
Type: **string**
Description: If the source Zip archive was created with a password, provide it in this parameter.

Returns

Parameter: **Status**
Type: **Object**
Description: The status object is used to determine when the zip procedure is complete. The zip procedure is launched in a new thread (process) so that other events may run during lengthy zip/unzip procedures. Use the .Finished property of this object to determine when the operation is complete. The Status object has these usable properties:

- **Finished** - Returns True/False status letting your script or application know when the operation is finished.
- **ZipError** - Returns a string. An empty string value indicates no problem. A value in ZipError usually indicates an error condition, but it may also be used for informational warnings about the zip/unzip procedure performed.
- **ThreadID** - Returns an integer value of the thread ID of the zip/unzip operation. This is used for diagnostic purposes. A value of -1 indicates that the thread ID has not yet been assigned.

Example

```
Sub Main()

    Dim ZO, zSrc, zDest

    zSrc = "D:\Backup\MyData.zip"
    zDest = "C:\DATA"

    Set ZO = hs.UnZip(zSrc, zDest, False, True, "")

    If Not IsObject(ZO) Then
        hs.WriteLog "Nothing","---------- It is not an object."
        Exit Sub
    End If

    Do While ZO.Finished = False
        hs.WriteLog "Zip_UnZip","Waiting on thread " & CStr(ZO.ThreadID)
        hs.WaitSecs 2
    Loop

    If Len(Trim(ZO.ZipError)) = 0 Then
        hs.WriteLog "Zip_UnZip","Done with the UnZip function - there were no errors or wa
    Else
        hs.WriteLog "Zip_UnZip","Done with the UnZip function - there was this error or wa
    End If

End Sub
```

See Also

- Serial Port Communication
- Network Information
- GetOSVersion
- RecurseFiles
- RecurseFilesEx
- RestartSystem
- Zip
- Keys

Zip

Purpose

This command will zip up files and create a zip archive file.

Parameters

Parameter: **ZipWhat**
Type: **string**
Description: Path to a directory, or path and filename of the directory or file to be added to a zip archive file.

Parameter: **ZipFileName**
Type: **string**
Description: Path and filename of the Zip archive file to be created or to have files added to.

Parameter: **compression** (optional)
Type: **integer (.NET Short)**
Description: The zip file compression level to use - the higher the level, the longer the zip operation will take. If this parameter is not provided, a default value of 6 is used. The valid values are from 0 to 9.

Parameter: **password** (optional)
Type: **string**
Description: If you wish the file(s) to be password protected in the archive, provide the password here. (Case sensitive)

Parameter: **RemoveBase** (optional)
Type: **boolean**
Description: If False, the entire directory structure up to and including the source file or directory will be included in the zip archive. If this parameter is not provided, then by default it is True and the directory structure before the starting point of the source files will be removed.

> Example: ZipWhat is C:\Program Files\HomeSeer\HTML\MyStuff (a directory) MyStuff has sub-directories Stuff1 and Stuff2.
>
> With this parameter True, the resulting archive will have Stuff1*.* and Stuff2*.* in it.

With this parameter False, the resulting archive will have Program Files\HomeSeer\HTML\MyStuff\Stuff1*.* and Program Files\HomeSeer\HTML\MyStuff\Stuff2*.* in it.

Parameter: **Flatten** (optional)
Type: **boolean**
Description: If True, the files will be put in the zip archive without any path information. Files that have the same filename but are in different subdirectories will result in only one of the files being left in the archive at the end of the Zip function.

Returns

Parameter: **Status**
Type: **Object**
Description: The status object is used to determine when the zip procedure is complete. The zip procedure is launched in a new thread (process) so that other events may run during lengthy zip/unzip procedures. Use the .Finished property of this object to determine when the operation is complete. The Status object has these usable properties:

- **Finished** - Returns True/False status letting your script or application know when the operation is finished.
- **ZipError** - Returns a string. An empty string value indicates no problem. A value in ZipError usually indicates an error condition, but it may also be used for informational warnings about the zip/unzip procedure performed.
- **ThreadID** - Returns an integer value of the thread ID of the zip/unzip operation. This is used for diagnostic purposes. A value of -1 indicates that the thread ID has not yet been assigned.

Example

```
Sub Main()

    Dim ZO, zSrc, zDest

    zSrc = "C:\DATA"
    zDest = "D:\Backup\MyData.zip"

    Set ZO = hs.Zip(zSrc, zDest, 6, "", True, False)


    If Not IsObject(ZO) Then
        hs.writelog "Nothing","-------- It is not an object."
        Exit Sub
    End If

    Do While ZO.Finished = False
        hs.WriteLog "Zip_UnZip","Waiting on thread " & CStr(ZO.ThreadID)
        hs.WaitSecs 2
    Loop

    hs.WriteLog "Zip_UnZip","Done With Zip Function."


End Sub
```

See Also

Serial Port Communication
Network Information
GetOSVersion
RecurseFiles
RecurseFilesEx
RestartSystem
UnZip
Keys

Home > Scripting > Computer > Keys

Keys

Sub Keys(ByVal KeyCode As String, ByVal Title As String, ByVal Wait As Boolean)

Purpose

This function allows you send keyboard commands to a running application. This is merely an interface into the .NET `SendKeys.Send` function.

Each key is represented by one or more characters. To specify a single keyboard character, use the character itself. For example, to represent the letter A use "A" for the string. To represent more than one character, append each additional character to the one preceding it. To represent the letters A, B, and C, use "ABC" for the string.

The plus sign (+), caret (^), percent sign (%), tilde (~), and parentheses ( ) have special meanings to the SendKeys function. To specify one of these characters, enclose it within braces ({}). For example, to specify the plus sign, use {+}. Brackets ([ ]) have no special meaning to SendKeys, but you must enclose them in braces. In other applications, brackets do have a special meaning that may be significant when dynamic data exchange (DDE) occurs. To specify brace characters, use {{} and {}}.

To specify characters that aren't displayed when you press a key, such as ENTER or TAB, and keys that represent actions rather than characters, use the codes shown below:

Key	Code
BACKSPACE	{BACKSPACE}, {BS}, or {BKSP}
BREAK	{BREAK}
CAPS LOCK	{CAPSLOCK}
DEL or DELETE	{DELETE} or {DEL}
DOWN ARROW	{DOWN}
END	{END}
ENTER	{ENTER}or ~
ESC	{ESC}
HELP	{HELP}
HOME	{HOME}
INS or INSERT	{INSERT} or {INS}
LEFT ARROW	{LEFT}
NUM LOCK	{NUMLOCK}
PAGE DOWN	{PGDN}
PAGE UP	{PGUP}
PRINT SCREEN	{PRTSC}
RIGHT ARROW	{RIGHT}
SCROLL LOCK	{SCROLLLOCK}
TAB	{TAB}
UP ARROW	{UP}
F1	{F1}
F2	{F2}
F3	{F3}
F4	{F4}
F5	{F5}
F6	{F6}
F7	{F7}
F8	{F8}
F9	{F9}
F10	{F10}
F11	{F11}
F12	{F12}
F13	{F13}
F14	{F14}
F15	{F15}
F16	{F16}

To specify keys combined with any combination of the SHIFT, CTRL, and ALT keys, precede the key code with one or more of the following codes:

Key	Code
SHIFT	+
CTRL	^
ALT	%

To specify that any combination of SHIFT, CTRL, and ALT should be held down while several other keys are pressed, enclose the code for those keys in parentheses. For example, to specify to hold down SHIFT while E and C are pressed, use "+(EC)". To specify to hold down SHIFT while E is pressed, followed by C without SHIFT, use "+EC".

To specify repeating keys, use the form {key number}. You must put a space between key and number. For example, {LEFT 42} means press the LEFT ARROW key 42 times; {h 10} means press H 10 times.

Note that you can't use SendKeys to send keystrokes to an application that is not designed to run in Microsoft Windows. Sendkeys also can't send the PRINT SCREEN key {PRTSC} to any application.

Parameters

Parameter: **KeyCode**
Type: **String**
Description: Is the key code to send (see below for special codes).

Parameter: **Title**
Type: **String**

Description: This is the title string that appears in the main window of the target application you wish to control.

Parameter: **Wait**
Type: **Boolean** (optional)
Description: This parameter is true to slow down the sending of the keys. Normally you want this to be TRUE, or (1).

Returns

None.

Example

This script will launch the calculator program:

```
sub main()

      dim I
      i=hs.launch("calc.exe","")

end sub
```

This script will use the calculator to add some numbers:

```
sub main()

   hs.speak "I will add some numbers"
   hs.keys "1{+}2{+}3{ENTER}","calc",1

end sub
```

See Also

Serial Port Communication
Network Information
GetOSVersion
RecurseFiles
RecurseFilesEx
RestartSystem
UnZip
Zip

Devices

In This Section

See Also

About Scripts
Applications and Plugins
Computer
Email
Events
Internet
Phone
Scripts
Speech Recognition
Strings, Global Variables, and Encryption
Time and Calendar
Text-To-Speech and Media

The Device Class

DeviceClass Properties and Procedures

Use caution when working with the DeviceClass properties directly. Internally, HomeSeer will compare, for example, an address from the device class to the address provided in a script command by making both lowercase or both uppercase. When you access the DeviceClass directly, you are getting the address exactly as it was entered by the user, so one device could have an address of "Hello" while another has an address of "HeLLo".

In MOST cases of accessing a property or procedure, there is a parameter of "hs" which is the type IHSApplication. This is the hs object itself. The reason for this is for data continuity. When you access the DeviceClass from a plug-in, a COPY of the DeviceClass object is what traverses the interface to the plug-in, and the plug-in is not accessing the real object. By including the hs object, you are indicating to HomeSeer that you want the latest information (GET) or are making a change (SET) and HomeSeer uses this reference to work with the actual DeviceClass object.

Example:

When you retrieve the location without providing a valid HomeSeer Interface Object (hs):

In your script or plug-in, you get a reference to the device that you want to work with (hs.GetDeviceByRef) and store the object in the variable dv. At the time you got that object reference, the location was "Family Room"

Time passes, and through the HomeSeer User Interface, somebody has changed the location of that device to "Den".

Now, you retrieve the location name, but you do not provide a valid HomeSeer Application Interface (hs) reference:

```
Dim Loc As String = dv.Location(Nothing)
```

If you look at the Loc variable, it will still be "Family Room". However, if you get the location and include the hs object:

```
Dim Loc As String = dv.Location(hs)
```

Now the Loc variable contains "Den", and this is because THROUGH the hs object, HomeSeer retrieved the information from the "Live" version of the object.

Reference

```
Public Property Ref(ByVal hs As IHSApplication) As Integer
```

The Ref property holds the device's unique device reference number. The Ref should never be changed except by a plug-in or script which has first used a procedure to generate a Ref that is guaranteed to be unique in the system.

```
Public Property Address(ByVal hs As IHSApplication) As String
```

The Address is a user or plug-in assigned string that identifies the device within a logical grouping. When you GET the value of this property, it always returns the Address AND the Code separated by a hiphen. For example, if the Address were set to HELLO, and the Code were set to WORLD, retrieving the Address would result in the string "HELLO-WORLD". The Code is always set separate from the Address. This field might be used to identify the module in a machine for which there are several sub-points, and each sub-point is a different Code - as such, all of the members of the module would be given the same Address and unique Code values.

```
Public Property Code(ByVal hs As IHSApplication) As String
```

The Code is treated both separately and in combination with the Address property. Both GET and SET may be done on the Code, but when a GET is done on the Address, the string returned is in the format Address-Code, with the value in this property being the Code. For example, a Z-Wave device that is a part of the network 00AABBCC and is Node 6 might have 5 child devices, so each device would have an Address of 00AABBCC-6, but a unique Code such as Q01, Q02, Q03 such that any one of the devices may have a full address of 00AABBCC-6-Q02

Identity

```
Public Property Name(ByVal hs As IHSApplication) As String
```

The Name property holds the name of the device, such as "Light", "Lamp", or "Heater"

```
Public Property Location(ByVal hs As IHSApplication) As String
```

This is the location name of the device, such as "Family Room".

```
Public Property Location2(ByVal hs As IHSApplication) As String
```

This is a second location modifier, which may be disabled in the HomeSeer settings, or if used can be used to further qualify the location of a device such as "First Floor".

Public Property UserNote(ByVal hs As IHSApplication) As String

This property stores any information the user so chooses, and is also editable on the device management page by clicking on the note icon.

Status

Public ReadOnly Property devString(ByVal hs As IHSApplication) As String

This is the device string for the device. When this property contains a value, it can override the display of the device's normal status display which is based upon the device's value. This property may contain HTML if HTML features are desired to be used when the device is viewed on the device utility page or the status views. This property is Read Only, so script commands must be used to modify the string value such as hs.SetDeviceString

Public ReadOnly Property devValue(ByVal hs As IHSApplication) As Double

This is the device's numerical value, which can be positive or negative and may contain a decimal point. Setting the device's value can cause changes to occur or change the status of the device. This property is Read Only, as the device value needs to be changed using commands such as hs.SetDeviceValue

Public Property Last_Change(ByVal hs As IHSApplication) As Date

This is the date and time that the device's value or string was last updated. Some ways of updating the value or string may explicitly block this from being updated, but in most cases it reflects the date and time of the last change.

Configuration

Public Property Device_Type_String(ByVal hs As IHSApplication) As String

The actual device type of a device is determined by information in the DeviceTypeInfo object (See DeviceType_Get and DeviceType_Set). This property may be used to hold a more "user friendly" device type string which is displayed on the device utility page. For example, if the device is owned by the Z-Wave plug-in, the DeviceTypeInfo object may identify it as a plug-in device type, but this property might display "Z-Wave Switch Multilevel".

Public ReadOnly Property DeviceType_Get(ByVal hs As IHSApplication) As DeviceTypeInfo
Public WriteOnly Property DeviceType_Set(ByVal hs As IHSApplication) As DeviceTypeInfo

The DeviceTypeInfo object holds several pieces of information describing the device type of the device. If the device is used with a technology API such as a Thermostat, Media, or Security, then the DeviceTypeInfo specifically identifies which part of the API the device fulfills.

Public Property Status_Support(ByVal hs As IHSApplication) As Boolean

This property indicates (when True) that the device supports the retrieval of its status on-demand through the "Poll" feature on the device utility page. The plug-in which owns the device is responsible for returning the status when the poll command is issues.

Public Property Can_Dim(ByVal hs As IHSApplication) As Boolean

This property is largely unused in HS3. When set to True and no other device value/status pairs have been assigned to the device, the default value/status pairs assigned will allow for levels/values from 1 to 99 in addition to 0 (Off) and 100 (On).

Public Property Image(ByVal hs As IHSApplication) As String

The Image property holds a path string to an image file to represent the device on the status views pages. The image path should be referenced from the root of the HTML folder under the main HomeSeer folder.

Public Property Interface(ByVal hs As IHSApplication) As String

This property holds the name of the plug-in that owns/manages this device. If the property is null or an empty string, the device is not managed by a plug-in.

Public Property InterfaceInstance(ByVal hs As IHSApplication) As String

This property holds the instance name of the plug-in that owns/manages this device. If the property is null or an empty string, either the plug-in does not support multiple instances (if the Interface property is not blank) or the device is not managed by a plug-in.

Public Property ScriptName(ByVal hs As IHSApplication) As String
Public Property ScriptFunc(ByVal hs As IHSApplication) As String

These properties are used ONLY when the Device_Type's API is set to Script, and the Device_Type's Device_Type is set to one of the script action values (See eDeviceType_Script) This provides functionality that will cause a script to be run when the device's value, string, or either are changed. ScriptName is the name of the script file to be run, and ScriptFunc is the name of a procedure in the script file to be called - if no ScriptFunc is provided, then Sub Main will be called.

When the script is run, it will be passed parameters as an object array, and those parameters are:

Parm(0) - **Integer** - The device reference ID.
Parm(1) - DeviceScriptChange (**Integer**) - Indicates what changed to cause the script to be run.

Parm(2) - **Double** - The device's new value.
Parm(3) - **String** - The device's new string.

Display

```
Public Property ScaleText(ByVal hs As IHSApplication) As String
```

A device that is used to display (for example) a temperature, the scale (Fahrenheit or Celsius) may not be known at the time the device is created or may be set/changed by an external device such that the device value/status pairs cannot be configured to display the proper scale symbol. To address this, plug-ins may update this property with the correct scale text just prior to adjusting the device's value. This property may be retrieved by other systems displaying this device's status and used in a similar manner to how it is used with the HomeSeer user interfaces.

```
Public Property AdditionalDisplayData(ByVal hs As IHSApplication) As String()
```

Similar to ScaleText, this property is used to enhance the device status display when variable elements of data are a part of the device status. For example, a Z-Wave enabled Smoke Detector may report an alarm, as well as location information. Since the variable location information cannot be assigned to value/status pairs in advance, this array of string values may be used.

Device Association

```
Public Sub AssociatedDevice_Add(ByVal hs As IHSApplication, ByVal dvRef As Integer)
Public ReadOnly Property AssociatedDevices_Count(ByVal hs As IHSApplication) As Integer
Public Sub AssociatedDevice_ClearAll(ByVal hs As IHSApplication)
Public ReadOnly Property AssociatedDevices(ByVal hs As IHSApplication) As Integer()
Public ReadOnly Property AssociatedDevices_List(ByVal hs As IHSApplication) As String
Public Sub AssociatedDevice_Remove(ByVal hs As IHSApplication, ByVal dvRef As Integer)

Public ReadOnly Property Parent As Enums.eRootChildStatus
Public ReadOnly Property Child As Enums.eRootChildStatus
```

These procedures and properties allow for getting information or making changes regarding the association of devices to one another. The typical usage is to associate one device (for example a Z-Wave Root Device) with several devices (Z-Wave Child Devices). For devices owned by plug-ins which represent technology API devices, it is strongly recommended for enumeration purposes that the single parent-multiple child relationship is used.

Associating devices should also be accompanied by the setting of the Root (Parent) device type on the parent device in a cluster of related devices. Each defined Device Type API contains a Device Type which indicates a Root device for that API, and a Device Type constant also exists to indicate a root device in the situation where there is a parent/child relationship between devices that do NOT belong to a specific technology API.

To add an association of another device to a device, use AssociatedDevice_Add. Example: dv.AssociatedDevice_Add(hs, 1234) - associates the device referenced by the device ID 1234 to the device class object dv.

To determine how many devices are associated to a device (in the device class object 'dv'), use AssociatedDevices_Count.

AssociatedDevice_ClearAll will remove all associated devices from the one which the AssociatedDevice_ClearAll procedure is called from.

AssociatedDevices and AssociatedDevices_List both return the device reference ID numbers for any devices associated with the device in which the property/procedure is called. AssociatedDevices returns an array of integers, and AssociatedDevices_List returns a comma separated string list.

AssociatedDevice_Remove will remove a single associated device reference number from the list of associated devices, which removes the association.

Relationship Status

```
Public Property Relationship(ByVal hs As IHSApplication) As Enums.eRelationship
```

The Relationship property can be used to determine or set the parent (root) or child status of a device. The return value is an Enum (eRelationship) which can indicate whether the device is a Parent/Root (it has child devices associated with it), a Child device (it is associated with a parent device), or Standalone (it is not associated with any other device). Additionally, the return value will indicate Not Set if the device has never had one of the values set to it, or Indeterminate, which may be used by a device in the process of being created or that could be in transition from one state to another. (Indeterminate is rarely used.)

Misc Bits - Check, Clear, Set

```
Public Function MISC_Check(ByVal hs As IHSApplication, ByVal Misc As Enums.dvMISC) As Boolean
Public Sub MISC_Clear(ByVal hs As IHSApplication, ByVal MISC As Enums.dvMISC)
Public Sub MISC_Set(ByVal hs As IHSApplication, ByVal MISC As Enums.dvMISC)
```

These procedures allow you to determine if various bits in the device's MISC settings are set or not, or to make changes to those bit settings. All of these procedures are aided by the use of an Enum called dvMISC so that more friendly names may be used instead of odd numerical values. The list of dvMISC values may be viewed here.

MISC_Check is used to determine if the selected MISC bit is Set (Returns True) or not set/cleared (Returns False).
Example: `If dv.MISC_Check(hs, Enums.dvMISC.NO_LOG) Then` can be used to determine if the NO_LOG option was set.

MISC_Set and MISC_Clear are used to Set/Enable or Reset/Clear the indicated bit respectively.

See Also

- Device Exists, Reference, Address and/or Code
- Creating, Deleting, or Accessing Devices
- Device Value, String, or Last Change
- Device Energy Management
- Device Control API (CAPI)

dvMISC

This Enum holds values referencing individual bits in an integer which indicate different characteristics of a device.

```
Enum dvMISC As UInteger
        NO_LOG = 8                      ' No logging to the log for this device
        STATUS_ONLY = &H10              ' Device cannot be controlled
        HIDDEN = &H20                   ' Device is hidden from the device utility page when
        '                                   Hide Marked is used.
        INCLUDE_POWERFAIL = &H80        ' The device's state is restored if power failure
        '                                   recovery is enabled
        SHOW_VALUES = &H100             ' If not set, device control options will not be
displayed.
        AUTO_VOICE_COMMAND = &H200      ' When set, this device is included in the voice
recognition
        '                                   context for device commands.
        VOICE_COMMAND_CONFIRM = &H400   ' When set, voice commands for this device are
confirmed.
        NO_STATUS_TRIGGER = &H20000     ' If set, the device status values will not appear
        '                                   in the device change trigger.
        CONTROL_POPUP = &H100000        ' The controls for this device should appear in a popup
        '                                   window on the device utility page.
End Enum
```

See Also

- eRelationship
- DeviceScriptChange
- Device Value Status Pairs
- Device Value Graphic Pairs
- Device Type
- Device_Type_String

eRelationship

This eNum is used as the return for the Parent and Child properties of the DeviceClass object, and are as follows:

```
Enum eRelationship As Integer
   Not_Set = 0
   Indeterminate = 1   ' Could not be determined
   Parent_Root = 2
   Standalone = 3
```

```
    Child = 4
End Enum
```

See Also

dvMISC
DeviceScriptChange
Device Value Status Pairs
Device Value Graphic Pairs
Device Type
Device_Type_String

DeviceScriptChange

This Enum is used when the Device_Type API is set to Script, and the Device_Type type is set to one of the script run values (See eDeviceType_Script). This Enum is one of the parameters passed to the script that is run when the device changes, and it indicates what changed to cause the script to be run. The values are:

```
Enum DeviceScriptChange As Integer
   DevValue = 1     ' The device value changed.
   DevString = 2     ' The device string changed.
   Both = 3      ' Both the device value and string changed.
End Enum
```

See Also

dvMISC
eRelationship
Device Value Status Pairs
Device Value Graphic Pairs
Device Type
Device_Type_String

Device Value Status Pairs

Devices hold a value property (double) that represents the status in the device, and a string which can be displayed regardless of the device value. It's possible to assign name->value pairs to a device. When this is done, the list of names is presented to the user in a drop list or some other UI form. Also, all trigger and actions dialogs will present the user with the value options rather than prompting them to enter a number. Strings are not as powerful as value/status pairs for control options on a device, but they are useful when strings are not known at device creation time or are dynamic during the runtime of HomeSeer.

Value/Status pairs can represent a status-only value, a control-only value, or both. An example would be the desire to have the value 100 represent "On" as a status, but a different value such as 200 with the status "Turn It On" for control. This arrangement allows a script or plug-in to trigger on the change of the device value to 200 which indicates a change needs to be made, and then set the value to 100 to indicate that the change is complete.

The sections of this help file under this topic will inform you about the VSPair object type, as well as the HomeSeer script interface commands which allow you to make changes to the value/status pairs.

See Also

dvMISC
eRelationship
DeviceScriptChange
Device Value Graphic Pairs
Device Type
Device_Type_String

VSPair

This is the VSPair object, which is used to describe a single value/status relationship or a range of values and associated status relationship. Multiples of these objects can be associated with a device to handle different types of control or status operations. Most of the modification of these pairs is done using the HomeSeer scripting/application interface commands that start with DeviceVSP_

```
Public Class VSPair

    Public PairType As VSVGPairType
    Public Render_Location As Enums.CAPIControlLocation
    Public RangeStart As Double
    Public RangeEnd As Double
    Public RangeStatusPrefix As String = ""
    Public RangeStatusSuffix As String = ""
    Public RangeStatusDecimals As Integer = 0
    Public RangeStatusDivisor As Double = 0
    Public IncludeValues As Boolean = True
    Public ValueOffset As Double = 0
    Public HasAdditionalData As Boolean = False
    Public HasScale As Boolean = False
    Public Const ScaleReplace As String = "@S@"
    Public Shared Function AddDataReplace(ByVal Index As Integer) As String
    Public ReadOnly Property ControlStatus As ePairStatusControl
    Public Property Render As Enums.CAPIControlType
    Public WriteOnly Property Status As String
    Public Property Value As Double
    Public Property StringList As String()
    Public WriteOnly Property StringListAdd As String

End Class
```

The definition for each member is as follows:

Name	Structure Member	Description
PairType		This enum indicates whether the pair represents a single value or a range of values.
Render_Location.	Row	If this is a control pair that is set to be rendered as a button, then set this to the row number to position the button at. Row or Column of 0 results in the button not being drawn, but the control option still exists.
Render_Location.	Column	If this is a control pair that is set to be rendered as a button, then set this to the column number to position the button at. Row or Column of 0 results in the button not being drawn, but the control option still exists.
Render_Location.	ColumnSpan	For more exact positioning of rendered controls on a device, you may indicate that a rendered control is to use multiple columns, thus allowing for alignment options. Note that not all 3rd party User Interfaces will be able to honor Row, Column, and ColumnSpan settings.
RangeStart		If this VSPair is a range, this contains the lowest value of the range being specified.
RangeEnd		If this VSPair is a range, this contains the highest value of the range being specified.
Value		If this VSPair is a single value pair and not a range, then this holds the value that this pair represents.

RangeStatusPrefix RangeStatusSuffix		These contain strings of text to be prepended (prefix) or appended (suffix) to the status string value as it is generated for a range value/status pair. These are not used for single-value pairs. An example of their use is in the range 1 to 99 to represent dim values, the prefix would be set to "Dim " and the suffix to "%", for a net status string when the value is 49 of "Dim 49%".
Status		When the value/status pair is not a range, this holds the status string to be displayed when the device is at the value set by the Value property.
RangeStatusDecimals		For range type value/status pairs, you may set this to a value > 0 to have that many decimal places displayed in the value. For example, if the range is 1 to 10, and the RangeStatusDecimals is set to 1, then the full range would encompass values such as: 1.0, 1.1, 1.2, 1.3.... 9.8, 9.9, 10.
RangeStatusDivisor		For range type value/status pairs, it may be inconvenient to modify the value so that it fits a more user-friendly display without messing up what the user has to enter for device value triggers - in that scenario, if you force the value to be 100 to represent 100K, the user may think they can enter 100 for a trigger when they need to enter 100000. To deal with this, set this property to 1000 and HomeSeer will divide the value by 1000 prior to formatting the display status string - the actual value will not be changed.
IncludeValues		For range value/status pairs, it is sometimes inconvenient having the value as part of the status when it is not indicative of anything meaningful. If IncludeValues is set to False, the status string generated will not include the values. Example: For a device which has an invalid state on values in the range 101 to 254, turn IncludeValues off (set it False) and set your RangeStatusPrefix to "INVALID VALUE", and that will cause HomeSeer to display INVALID VALUE for each of those values without having to use single value, value-status pairs.
ValueOffset		When it is desireable to have separate status and control value/status pair ranges, this property can be used to facilitate that since two separate status and control pairs cannot be for the same value. To use this, establish one range to use the true values of 1 to 100. Now, establish a second range to use the "fake" values of 101 to 200, but set the ValueOffset to 100, which causes HomeSeer to use a display status string with the value having 100 subtracted from it. For example, if you have a status range pair which creates a status of "Setting Is Currently 50" with a value of 50, you can have a control pair that creates a control option of "Set to 50 Degrees", which corresponds to the value 150. When a script or plug-in receives notification of the device changing to 150, the appropriate command can be sent to invoke the change to 50, and then the device may be set to 50 to indicate that the change has been made.
HasScale ScaleReplace		At the time a device is created, it may not be known whether its scale is meters or Miles, Fahrenheit or Celcius, or some other set of multiple scales. To help with those situations, set HasScale to True, use the constant ScaleReplace in your range prefix or suffix, and then at runtime when the device is being updated to a new value, set the device ScaleText to your scale (e.g. "degF" or "degC"), set the value, and then when the status is requested, HomeSeer will replace ScaleReplace (@S@) with your ScaleText.
HasAdditionalData AddDataReplace		The usage of these two properties is identical to that of HasScale/ScaleReplace except that you can use a virtually limitless number of replacements in the status string. To use, set HasAdditionalData to True, and then when the device is updated, set the device's AdditionalDisplayData (array of string) to the values that you want replaced. AddDataReplace is a shared (constant) function that can be used to generate a replacement variable for any number - for example, AddDataReplace with an argument value of 4 will return a replacement variable of "@%4@" (without quotes). If your range prefix or suffix contains this string (which you can use by calling AddDataReplace(4) when you are setting up the prefix or suffix), then that string will be replaced with the 4th value from the array of strings set on the device's AdditionalDisplayData property. e.g.: pair.HasAdditionalData = True pair.RangeStatusSuffix = AddDataReplace(0) & " of " & AddDataReplace(1) (Device value changes and becomes 5555...) Dim AddData(1) As String AddData(0) = "Miles" AddData(1) = "Asphalt" dv.AdditionalDisplayData(hs) = AddData hs.SetDeviceValue(MyDevice, 5555) Result: 5555 Miles of Asphalt
ControlStatus		This read-only property allows you to retrieve an Enum value indicating whether the pair is designated as being for status only, control only, or both status and control. Use the hs.DeviceVSP_ChangePair (and other functions) to set or change the ControlStatus.

Render		This property is used to get or set how the control status/value pair is to be rendered when control options are offered by a user interface. See the values under the CAPIControl/CAPIControlType topic.
StringList StringListAdd		When the UI Render type is set to a drop-down list of strings, set StringList to the array of string values to be displayed. You may add the items one at a time using StringListAdd, or may set them all at once using StringList.

See Also

DeviceVSP Methods

VSVGPairType

The VSVGPairType, used with both value/status and value/graphic pairs, is an Enum as follows:

```
Public Enum VSVGPairType
      SingleValue = 1
      Range = 2
End Enum
```

See Also

ePairStatusControl

ePairStatusControl

ePairStatusControl is an Enum used in value/status pairs and has the following values:

```
Public Enum ePairStatusControl
   Status = 1
   Control = 2
   Both = 3
End Enum
```

See Also

VSVGPairType

DeviceVSP Methods

Body of text here

See Also

VSPair

DeviceVSP_AddPair

To add the name->value pair to a device use this function:

```
Public Function DeviceVSP_AddPair(ByVal dvRef As Integer, ByVal Pair As VSPair) As Boolean
```

hs.DeviceVSP_AddPair(ref, Pair)

Where:

ref= device reference #

Pair = The VSPair object that you want added

You can check the return value (Boolean) to determine if it was successful or not.

See Also

DeviceVSP_ChangePair
DeviceVSP_CountAll
DeviceVSP_CountStatus
DeviceVSP_CountControl
DeviceVSP_ClearAll
DeviceVSP_ClearAny
DeviceVSP_ClearStatus
DeviceVSP_ClearControl
DeviceVSP_ClearBoth
DeviceVSP_Get
DeviceVSP_GetStatus
DeviceVSP_PairsProtected

DeviceVSP_ChangePair

This will change the pair type of an existing value/status pair.

```
Public Function DeviceVSP_ChangePair(ByVal dvRef As Integer, _
                                      ByVal Existing As VSPair, _
                                      ByVal NewType As ePairStatusControl) As Boolean
```

hs.DeviceVSP_ChangePair(ref, Existing, NewType)

Where:

ref= device reference #

Existing = The current VSPair (value/status pair) object that is set on the device.

NewType = The new ePairStatusControl type (Status, Control, Both) that you want the pair type changed to.

For example, to change the pair type from Status to Both:

```
hs.DeviceVSP_ChangePair(ref, Pair, ePairStatusControl.Both)
```

You can check the return value (Boolean) to determine if it was successful or not.

Note: Currently there is a feature in HS3 being evaluated to determine if it will stay, which is the Protection mode of a value/status pair which would normally prevent this from working if the protection

mode was not Off or Do Not Delete. At this time, DeviceVSP_ChangePair overrides this and forces the protection to be set to OFF.

See Also

DeviceVSP_AddPair
DeviceVSP_CountAll
DeviceVSP_CountStatus
DeviceVSP_CountControl
DeviceVSP_ClearAll
DeviceVSP_ClearAny
DeviceVSP_ClearStatus
DeviceVSP_ClearControl
DeviceVSP_ClearBoth
DeviceVSP_Get
DeviceVSP_GetStatus
DeviceVSP_PairsProtected

DeviceVSP_CountAll

Use this function to get a count of all value/status pairs on a device.

```
Public Function DeviceVSP_CountAll(ByVal dvRef As Integer) As Integer
```

Count = hs.DeviceVSP_CountAll(ref)

Where:

ref= device reference ID

You can check the return value (Integer) to determine if it was successful or not. Return values less than zero (0) indicate an error condition such as the device reference ID being invalid.

See Also

DeviceVSP_AddPair
DeviceVSP_ChangePair
DeviceVSP_CountStatus
DeviceVSP_CountControl
DeviceVSP_ClearAll
DeviceVSP_ClearAny
DeviceVSP_ClearStatus
DeviceVSP_ClearControl
DeviceVSP_ClearBoth
DeviceVSP_Get
DeviceVSP_GetStatus
DeviceVSP_PairsProtected

DeviceVSP_CountStatus

Use this function to get a count of only the status type value/status pairs on a device.

```
Public Function DeviceVSP_CountStatus(ByVal dvRef As Integer) As Integer
```

Count = hs.DeviceVSP_CountStatus(ref)

Where:

ref= device reference ID

You can check the return value (Integer) to determine if it was successful or not. Return values less than zero (0) indicate an error condition such as the device reference ID being invalid.

See Also

DeviceVSP_AddPair
DeviceVSP_ChangePair
DeviceVSP_CountAll
DeviceVSP_CountControl
DeviceVSP_ClearAll
DeviceVSP_ClearAny
DeviceVSP_ClearStatus
DeviceVSP_ClearControl
DeviceVSP_ClearBoth
DeviceVSP_Get
DeviceVSP_GetStatus
DeviceVSP_PairsProtected

DeviceVSP_CountControl

Use this function to get a count of all control type value/status pairs on a device.

```
Public Function DeviceVSP_CountControl(ByVal dvRef As Integer) As Integer
```

```
Count = hs.DeviceVSP_CountControl(ref)
```

Where:

ref= device reference ID

You can check the return value (Integer) to determine if it was successful or not. Return values less than zero (0) indicate an error condition such as the device reference ID being invalid.

See Also

DeviceVSP_AddPair
DeviceVSP_ChangePair
DeviceVSP_CountAll
DeviceVSP_CountStatus
DeviceVSP_ClearAll
DeviceVSP_ClearAny
DeviceVSP_ClearStatus
DeviceVSP_ClearControl
DeviceVSP_ClearBoth
DeviceVSP_Get
DeviceVSP_GetStatus
DeviceVSP_PairsProtected

DeviceVSP_ClearAll

Use this function to CLEAR all value/status pairs from a device.

```
Public Sub DeviceVSP_ClearAll(ByVal dvRef As Integer, ByVal TrueConfirm As Boolean)
```

```
hs.DeviceVSP_ClearAll(ref, True)
```

Where:

ref= device reference ID

True = The constant True or a variable indicating True must be passed as the second parameter as confirmation that you wish this to take place.

See Also

DeviceVSP_AddPair
DeviceVSP_ChangePair
DeviceVSP_CountAll
DeviceVSP_CountStatus
DeviceVSP_CountControl
DeviceVSP_ClearAny
DeviceVSP_ClearStatus
DeviceVSP_ClearControl
DeviceVSP_ClearBoth
DeviceVSP_Get
DeviceVSP_GetStatus
DeviceVSP_PairsProtected

DeviceVSP_ClearAny

This will clear any (control, status, or both) value/status pair out of the device that matches the given value parameter.

```
Public Function DeviceVSP_ClearAny(ByVal dvRef As Integer, ByVal Value As Double) As Boolean
```

Success = hs.DeviceVSP_ClearAny(ref, Value)

Where:

ref= device reference #

Value = The value of the value/status pair you wish removed.

You can check the return value (Boolean) to determine if it was successful or not.

See Also

DeviceVSP_AddPair
DeviceVSP_ChangePair
DeviceVSP_CountAll
DeviceVSP_CountStatus
DeviceVSP_CountControl
DeviceVSP_ClearAll
DeviceVSP_ClearStatus
DeviceVSP_ClearControl
DeviceVSP_ClearBoth
DeviceVSP_Get
DeviceVSP_GetStatus
DeviceVSP_PairsProtected

DeviceVSP_ClearStatus

This will clear any status type value/status pair out of the device that matches the given value parameter.

```
Public Function DeviceVSP_ClearStatus(ByVal dvRef As Integer, ByVal Value As Double) As Boolean
```

Success = hs.DeviceVSP_ClearStatus(ref, Value)

Where:

ref= device reference #

Value = The value of the status type value/status pair you wish removed.

You can check the return value (Boolean) to determine if it was successful or not.

See Also

DeviceVSP_AddPair
DeviceVSP_ChangePair
DeviceVSP_CountAll
DeviceVSP_CountStatus
DeviceVSP_CountControl
DeviceVSP_ClearAll
DeviceVSP_ClearAny
DeviceVSP_ClearControl
DeviceVSP_ClearBoth
DeviceVSP_Get
DeviceVSP_GetStatus
DeviceVSP_PairsProtected

DeviceVSP_ClearControl

This will clear any control type value/status pair out of the device that matches the given value parameter.

```
Public Function DeviceVSP_ClearControl(ByVal dvRef As Integer, ByVal Value As Double) As Boolean
```

Success = hs.DeviceVSP_ClearControl(ref, Value)

Where:

ref= device reference #

Value = The value of the control type value/status pair you wish removed.

You can check the return value (Boolean) to determine if it was successful or not.

See Also

DeviceVSP_AddPair
DeviceVSP_ChangePair
DeviceVSP_CountAll
DeviceVSP_CountStatus
DeviceVSP_CountControl
DeviceVSP_ClearAll
DeviceVSP_ClearAny
DeviceVSP_ClearStatus
DeviceVSP_ClearBoth
DeviceVSP_Get
DeviceVSP_GetStatus
DeviceVSP_PairsProtected

DeviceVSP_ClearBoth

This will clear any "Both" type value/status pair out of the device that matches the given value parameter.

```
Public Function DeviceVSP_ClearBoth(ByVal dvRef As Integer, ByVal Value As Double) As Boolean
```

Success = hs.DeviceVSP_ClearBoth(ref, Value)

Where:

ref= device reference #

Value = The value of the "Both" (status and control) type value/status pair you wish removed.

You can check the return value (Boolean) to determine if it was successful or not.

See Also

DeviceVSP_AddPair
DeviceVSP_ChangePair
DeviceVSP_CountAll
DeviceVSP_CountStatus
DeviceVSP_CountControl
DeviceVSP_ClearAll
DeviceVSP_ClearAny
DeviceVSP_ClearStatus
DeviceVSP_ClearControl
DeviceVSP_Get
DeviceVSP_GetStatus
DeviceVSP_PairsProtected

DeviceVSP_Get

This will retrieve a value/status pair object (VSPair) from a device if it matches the value and type provided.

```
Public Function DeviceVSP_Get(ByVal dvRef As Integer, _
                              ByVal Value As Double, _
                              ByVal VSPType As ePairStatusControl) As VSPair
```

```
MyPair = hs.DeviceVSP_Get(ref, Value, VSPType)
If MyPair Is Nothing Then
   hs.WriteLog("Error","Could not find the value/status pair for the value " & Value.ToString & " on the device " & hs.DeviceName(ref))
   Exit Sub
End If
```

Where:

ref= device reference #

Value = The value of the value/status pair you are looking for (use the starting value of the range to retrieve a range type pair).

VSPType = The ePairStatusControl type (Status, Control, Both) that you are looking for.

You can check the return value (VSPair object) to determine if it was successful or not. If the returned object = Nothing, then the pair matching the provided parameters was not found.

See Also

DeviceVSP_AddPair
DeviceVSP_ChangePair
DeviceVSP_CountAll
DeviceVSP_CountStatus
DeviceVSP_CountControl
DeviceVSP_ClearAll
DeviceVSP_ClearAny
DeviceVSP_ClearStatus
DeviceVSP_ClearControl
DeviceVSP_ClearBoth
DeviceVSP_GetStatus
DeviceVSP_PairsProtected

DeviceVSP_GetStatus

This will retrieve a status string given a specific value.

```
Public Function DeviceVSP_GetStatus(ByVal dvRef As Integer, _
                                    ByVal Value As Double, _
                                    ByVal VSPType As ePairStatusControl) As String
```

MyStatus = hs.DeviceVSP_GetStatus(ref, Value, VSPType)

Where:

ref= device reference #

Value = The value of the value/status pair you are looking for (use any value of the range to retrieve a range type status).

VSPType = The ePairStatusControl type (Status, Control, Both) that you are looking for. If the type is Status or Control, the string returned will be the formatted status or control string for the value given. If the type is Both, then the string returned will be in the form: Status: (text), Control: (text)

See Also

DeviceVSP_AddPair
DeviceVSP_ChangePair
DeviceVSP_CountAll
DeviceVSP_CountStatus
DeviceVSP_CountControl
DeviceVSP_ClearAll
DeviceVSP_ClearAny
DeviceVSP_ClearStatus
DeviceVSP_ClearControl
DeviceVSP_ClearBoth
DeviceVSP_Get
DeviceVSP_PairsProtected

DeviceVSP_PairsProtected

To check to see whether the device Value/Status pairs are protected, which applies to devices owned by plug-ins, use this function:

```
Function DeviceVSP_PairsProtected(ByVal dvRef As Integer) As Boolean
```

PStatus = hs.DeviceVSP_PairsProtected(ref)

Where:

ref= device reference #

You can check the return value (Boolean) to determine if the pairs are protected from editing in the HomeSeer UI by the user.

See Also

DeviceVSP_AddPair
DeviceVSP_ChangePair
DeviceVSP_CountAll
DeviceVSP_CountStatus
DeviceVSP_CountControl
DeviceVSP_ClearAll
DeviceVSP_ClearAny
DeviceVSP_ClearStatus
DeviceVSP_ClearControl
DeviceVSP_ClearBoth
DeviceVSP_Get
DeviceVSP_GetStatus

Device Value Graphic Pairs

Devices hold a value property (float) that is normally used to hold the dim level of X10 devices. It may be desirable to use the value to represent status in the device. It's possible to assign graphic->value pairs to a device. When this is done, the status as displayed for the device will include the graphic matching the device's current value.

To assign the graphic->value pairs to a device, use the function call hs.DeviceVGP_AddPair. To use the function, call:

hs.DeviceVGP_AddPair(dvRef, Pair)

Where:

dvRef = device reference # to set values to

Pair = graphic value pairs formatted in the class VGPair

Note

- The setting is saved in the configuration database so the call only needs to be made during some initialization.

See Also

dvMISC
eRelationship
DeviceScriptChange
Device Value Status Pairs
Device Type
Device_Type_String

VGPair

This is the VGPair object, which is used to describe a single value/graphic relationship or a range of values and associated graphics relationship. Multiples of these objects can be associated with a device to handle different types of graphics to represent different states (values) of a device. Most of the modification of these pairs is done using the HomeSeer scripting/application interface commands that start with DeviceVGP_

```
Public Class VGPair

    Public PairType As VSVGPairType
    Public RangeStart As Double
    Public RangeEnd As Double
    Public WriteOnly Property Graphic As String
    Public ReadOnly Property Value As Double
    Public WriteOnly Property Set_Value As Double

End Class
```

The definition for each member is as follows:

Name	Description
PairType	This enum indicates whether the pair represents a single value or a range of values.
RangeStart	If this VGPair is a range, this contains the lowest value of the range being specified.
RangeEnd	If this VGPair is a range, this contains the highest value of the range being specified.
Value (Read Only) Set_Value (Write Only)	If this VGPair is a single value pair and not a range, then this holds the value that this pair represents.
Graphic (Write Only)	This contains the path, relative to the HomeSeer HTML directory or absolute if outside the HTML directory.

See Also

DeviceVGP Methods

VSVGPairType

The VSVGPairType, used with both value/status and value/graphic pairs, is an Enum as follows:

```
Public Enum VSVGPairType
      SingleValue = 1
      Range = 2
End Enum
```

See Also

DeviceVGP Methods

Body of text here

See Also

VGPair

DeviceVGP_AddPair

To add the graphic->value pair to a device use this function:

```
Public Function DeviceVGP_AddPair(ByVal dvRef As Integer, ByVal Pair As VGPair) As Boolean
```

hs.DeviceVGP_AddPair(ref, Pair)

Where:

ref= device reference #

Pair = The VGPair object that you want added

You can check the return value (Boolean) to determine if it was successful or not.

See Also

DeviceVGP_Count
DeviceVGP_ClearAll
DeviceVGP_Clear
DeviceVGP_Get
DeviceVGP_GetGraphic
DeviceVGP_PairsProtected

DeviceVGP_Count

Use this function to get a count of all value/graphic pairs on a device.

```
Public Function DeviceVGP_Count(ByVal dvRef As Integer) As Integer
```

```
Count = hs.DeviceVGP_Count(ref)
```

Where:

ref= device reference ID

You can check the return value (Integer) to determine if it was successful or not. Return values less than zero (0) indicate an error condition such as the device reference ID being invalid.

See Also

DeviceVGP_AddPair
DeviceVGP_ClearAll
DeviceVGP_Clear
DeviceVGP_Get
DeviceVGP_GetGraphic
DeviceVGP_PairsProtected

DeviceVGP_ClearAll

Use this function to CLEAR all value/graphic pairs from a device.

```
Public Sub DeviceVGP_ClearAll(ByVal dvRef As Integer, ByVal TrueConfirm As Boolean)
```

```
hs.DeviceVGP_ClearAll(ref, True)
```

Where:

ref= device reference ID

True = The constant True or a variable indicating True must be passed as the second parameter as confirmation that you wish this to take place.

See Also

DeviceVGP_AddPair
DeviceVGP_Count
DeviceVGP_Clear
DeviceVGP_Get
DeviceVGP_GetGraphic
DeviceVGP_PairsProtected

DeviceVGP_Clear

This will clear any value/graphic pair out of the device that matches the given value parameter.

```
Public Function DeviceVGP_Clear(ByVal dvRef As Integer, ByVal Value As Double) As Boolean
```

```
Success = hs.DeviceVGP_Clear(ref, Value)
```

Where:

ref= device reference #

Value = The value of the value/graphic pair you wish removed.

You can check the return value (Boolean) to determine if it was successful or not.

See Also

DeviceVGP_AddPair
DeviceVGP_Count
DeviceVGP_ClearAll
DeviceVGP_Get
DeviceVGP_GetGraphic
DeviceVGP_PairsProtected

DeviceVGP_Get

This will retrieve a value/graphic pair object (VGPair) from a device if it matches the value provided.

```
Public Function DeviceVGP_Get(ByVal dvRef As Integer, _
                              ByVal Value As Double) As VGPair
```

```
MyPair = hs.DeviceVGP_Get(ref, Value)
If MyPair Is Nothing Then
   hs.WriteLog("Error","Could not find the value/graphic pair for the value " & Value.ToString & " on the device " & hs.DeviceName(ref))
   Exit Sub
End If
```

Where:

ref= device reference #

Value = The value of the value/graphic pair you are looking for (use the starting value of the range to retrieve a range type pair).

You can check the return value (VSPair object) to determine if it was successful or not. If the returned object = Nothing, then the pair matching the provided parameters was not found.

See Also

DeviceVGP_AddPair
DeviceVGP_Count
DeviceVGP_ClearAll
DeviceVGP_Clear
DeviceVGP_GetGraphic
DeviceVGP_PairsProtected

DeviceVGP_GetGraphic

This will retrieve a graphic path given a specific value.

```
Public Function DeviceVGP_GetGraphic(ByVal dvRef As Integer, _
                                     ByVal Value As Double) As String
```

MyGraphic = hs.DeviceVGP_GetGraphic(ref, Value)

Where:

ref= device reference #

Value = The value of the value/graphic pair you are looking for (use any value of the range to retrieve a range type graphic).

See Also

DeviceVGP_AddPair
DeviceVGP_Count
DeviceVGP_ClearAll
DeviceVGP_Clear
DeviceVGP_Get
DeviceVGP_PairsProtected

DeviceVGP_PairsProtected

To check to see whether the device Value/Graphic pairs are protected, which applies to devices owned by plug-ins, use this function:

```
Function DeviceVGP_PairsProtected(ByVal dvRef As Integer) As Boolean
```

PStatus = hs.DeviceVGP_PairsProtected(ref)

Where:

ref= device reference #

You can check the return value (Boolean) to determine if the pairs are protected from editing in the HomeSeer UI by the user.

See Also

DeviceVGP_AddPair
DeviceVGP_Count
DeviceVGP_ClearAll
DeviceVGP_Clear
DeviceVGP_Get
DeviceVGP_GetGraphic

Device Type

In previous versions of HomeSeer, the device type was a string property and a value that was used to describe the capabilities of the device. The string value was used to find a specific set of device capabilities, and HomeSeer would create a new device type (with a numerical suffix at the end) whenever it discovered that the device was modified from what the device type said it should have as capabilities.

In HomeSeer HS3, the high level meaning is very similar, but the functionality is different enough that NOTHING in reference to the previous versions should be re-used.

The Device Class object has two properties pertaining to device type descriptions:
Device_Type_String
and

DeviceType_Get (Read Only)
DeviceType_Set (Write Only)

Device_Type_String is a string value which has absolutely no bearing on functionality of the device. This string is what is displayed for the device type if that column of information is enabled on the device utility page. The value is accessed through the plug-in interface, which is a one-way interface requiring the passing of the HomeSeer application interface object when it is changed or when the most current value is retrieved.

DeviceType is an object (DeviceTypeInfo) with properties and procedures. The information stored is used in several of the APIs that HomeSeer supports to describe the device's role in the API. The DeviceType consists of three high-level items: The API designation, the device type, and the device sub-type. These (and the lower-level informational items) are described in the sub-topics to this entry.

See Also

dvMISC
eRelationship
DeviceScriptChange
Device Value Status Pairs
Device Value Graphic Pairs
Device_Type_String

DeviceTypeInfo Object

The DeviceType object (DeviceTypeInfo) is accessed directly from the device class object using the _Get and _Set properties. When a change is made to the DeviceTypeInfo object, _Set must be called to post the change to the device, and then hs.SaveEventsDevices should be called to force HomeSeer to save the device change.

The prototype for the class is:

```
Public Class DeviceTypeInfo

    Public Property Device_API As eDeviceAPI
    Public ReadOnly Property Device_API_Description As String
    Public Property Device_Type As Integer
    Public ReadOnly Property Device_Type_Description As String
    Public Property Device_SubType As Integer
    Public Property Device_SubType_Description As String

End Class
```

Example:

Retrieve the DeviceTypeInfo object:

```
Dim DT as DeviceAPI.DeviceTypeInfo = Nothing
DT = hs.DeviceType_Get(hs)
If DT IsNot Nothing Then
 ...
End If
```

Change the DeviceTypeInfo and Save the change

```
Dim DT as DeviceAPI.DeviceTypeInfo = Nothing

DT = hs.DeviceType_Get(hs)

If DT IsNot Nothing Then

 DT.Device_API = DeviceAPI.DeviceTypeInfo.eDeviceAPI.Plug_In
 hs.DeviceType_Set(hs) = DT
 hs.SaveEventsDevices
End If
```

See Also

Device_API

The Device_API property is an enum denoting the type of API, if any, that this device is a part of:

```
Public Property Device_API As DeviceAPI.DeviceTypeInfo.eDeviceAPI
```

The Device_API should be set appropriately if the device is a part of an API, set to Plug_In if it is not a part of an API but is owned by a Plug-In, or set to No_API if it is not owned by a plug-in and is not a part of an API.

Example:

```
Dim DT As New DeviceAPI.DeviceTypeInfo
DT.Device_API = DeviceAPI.DeviceTypeInfo.eDeviceAPI.Plug_In
DT.Device_Type = CInt(ntype)
dv.DeviceType_Set(hs) = DT
hs.SaveEventsDevices
```

See Also

- Device_API_Description (Read Only)
- Device_Type
- Device_Type_Description (Read Only)
- Device_SubType
- Device_SubType_Description

eDeviceAPI

```
<Serializable()> _
    Public Enum eDeviceAPI
        No_API = 0  ' All other devices.
        Plug_In = 4  ' Device is owned/managed by a plug-in.
        Thermostat = 16 ' Device is owned/managed by a plug-in and is a thermostat device.
        Media = 32  ' Device is owned/managed by a plug-in and is a media player device.
        Security = 8  ' Device is owned/managed by a plug-in and is a security device.
        SourceSwitch = 64 ' Device is owned/managed by a plug-in and is a matrix switch device.
        Script = 128  ' Device launches a script when the value and/or string changes.
    End Enum
```

See Also

Device_API_Description (Read Only)

The Device_API_Description read-only property is a string denoting the type of API, if any, that this device is a part of:

```
Public ReadOnly Property Device_API_Description As String
```

The Device_API determines what is returned by this property.

Example:

```
hs.WriteLog("Info", hs.DeviceName(dv.Ref) & " has a Device Type API of " &
hs.DeviceType_Get(hs).Device_API_Description)
```

See Also

- Device_API
- Device_Type
- Device_Type_Description (Read Only)
- Device_SubType
- Device_SubType_Description

Device_Type

The Device_Type property is an integer denoting the device type of an API, if any, that this device is a part of:

```
Public Property Device_Type As Integer
```

The Device_API should be set appropriately if the device is a part of an API, set to Plug_In if it is not a part of an API but is owned by a Plug-In, or set to No_API if it is not owned by a plug-in and is not a part of an API. If the Device_API is set to an API type such as Thermostat or Music, then the Device_Type should be set to one of the API types for those APIs. (See DeviceTypeInfo Enums)

Example:
(This sets the device to the thermostat API type "Mode Set")

```
Dim DT As New DeviceAPI.DeviceTypeInfo
DT.Device_API = DeviceAPI.DeviceTypeInfo.eDeviceAPI.Thermostat
DT.Device_Type = DeviceAPI.DeviceTypeInfo.eDeviceType_Thermostat.Mode_Set
dv.DeviceType_Set(hs) = DT
hs.SaveEventsDevices
```

Example:
(This sets the device to a plug-in custom type.)

```
Dim DT As New DeviceAPI.DeviceTypeInfo
DT.Device_API = DeviceAPI.DeviceTypeInfo.eDeviceAPI.Plug_In
DT.Device_Type = CInt(PlugDeviceType_X)
DT.Device_SubType = 4
dv.DeviceType_Set(hs) = DT
hs.SaveEventsDevices
```

See Also

- Device_API
- Device_API_Description (Read Only)
- Device_Type_Description (Read Only)
- Device_SubType
- Device_SubType_Description

eDeviceType_GenericRoot

Purpose

The eDeviceType_GenericRoot is not a device type like the other device types - it is a constant value integer (Value = 999) which is to be used when a device is to be a root device for a parent/child relationship, and does not fit any other API specific model.

Parameters

Parameter: **(Value)**
Type: **Integer**
Description: The value of this device type is 999.

See Also

eDeviceType_Media
eDeviceType_Plugin
eDeviceType_Script
eDeviceType_Security
eDeviceType_SourceSwitch
eDeviceType_Thermostat

eDeviceType_Media

```
Public Enum eDeviceType_Media
   Player_Status = 1
   Player_Status_Additional = 2
   Player_Control = 3
   Player_Volume = 4
   Player_Shuffle = 5
   Player_Repeat = 6
   Music_Genre = 7
   Music_Album = 8
   Music_Artist = 9
   Music_Track = 10
   Music_Playlist = 11
   Media_Type = 12
   Music_Selector_Control = 20 ' Used to track which instance of MusicAPI and selection mode (e.g. album, artists, playlists)
   Root = 99   ' Indicates a root device of a root/child grouping.
End Enum
```

See Also

eDeviceType_GenericRoot
eDeviceType_Plugin
eDeviceType_Script
eDeviceType_Security
eDeviceType_SourceSwitch
eDeviceType_Thermostat

eDeviceType_Plugin

The Plug-In device type indicates a device type that does NOT fit any of the API specific device types, but is a device type owned by a plug-in. The only defined Enum value is for indicating a Root device in a Parent(Root)/Child relationship.

```
Public Enum eDeviceType_Plugin
   Root = 99   ' Indicates a root device of a root/child grouping.
End Enum
```

See Also

eDeviceType_GenericRoot
eDeviceType_Media
eDeviceType_Script
eDeviceType_Security
eDeviceType_SourceSwitch
eDeviceType_Thermostat

eDeviceType_Script

```
Public Enum eDeviceType_Script
   Disabled = 0    ' Set the device type to this to temporarily stop scripts from being run.
   Run_On_Any_Change = 1  ' Set to this type to run the script on a value or string change.
   Run_On_Value_Change = 2  ' Set to this type to run the script when the value changes.
   Run_On_String_Change = 3 ' Set to this type to run the script when the string changes.
End Enum
```

See Also

eDeviceType_GenericRoot
eDeviceType_Media
eDeviceType_Plugin
eDeviceType_Security
eDeviceType_SourceSwitch
eDeviceType_Thermostat

eDeviceType_Security

```
Public Enum eDeviceType_Security
   Alarm = 1                  ' Alarm status & control (shows alarms that have occurred and
   '                          can also invoke an alarm - e.g. Duress)
   Arming = 10                ' Arming status & control (shows the state of the security arming and can set
arming state)
   Keypad = 20                ' Keypad status & control
   Zone_Perimeter = 30        ' A perimeter zone
   Zone_Perimeter_Delay = 31  ' A perimeter zone with a violation alarm delay
   Zone_Interior = 32         ' An interior zone (not normally armed in stay mode)
   Zone_Interior_Delay = 33   ' An interior zone (with a violation alarm delay when armed)
   Zone_Auxiliary = 34        ' An aux zone, not usually included in any arming mode
   Zone_Other = 35            ' A zone that does not fit any other zone description
   Zone_Safety_Smoke = 40     ' A smoke detector zone (not allowed to be bypassed)
   Zone_Safety_CO = 41        ' A Carbon Monoxide zone (not allowed to be bypassed)
   Zone_Safety_CO2 = 42       ' A Carbon Dioxide zone (not allowed to be bypassed)
   Zone_Safety_Other = 43     ' A zone for some other safety sensor that cannot be bypassed
   Output_Relay = 50          ' A general purpose output relay
   Output_Other = 51          ' A general purpose output (could be virtual as in a 'flag' output)
   Communicator = 60          ' Communicator status and (if available) control
   Siren = 70                 ' Siren output - status usually - control follows alarm state.
   Root = 99                  ' Indicates a root device of a root/child grouping.
End Enum
```

See Also

eDeviceType_GenericRoot
eDeviceType_Media
eDeviceType_Plugin
eDeviceType_Script
eDeviceType_SourceSwitch
eDeviceType_Thermostat

eDeviceType_SourceSwitch

```
Public Enum eDeviceType_SourceSwitch
   Invalid = 0
   System = 1              ' Indicates system status and/or contains system control capabilities.
   Source = 10             ' Indicates source status information and/or contains source control capabilities.
   Source_Extended = 15  ' An extension to Source, can be used for less common status or control features.
   Zone = 20               ' Indicates zone status information and/or contains zone control capabilities.
   Zone_Extended = 25    ' An extension to Zone, can be used for less common status or control features.
   Root = 99               ' The root device of a root/child grouping.
End Enum
```

See Also

eDeviceType_GenericRoot
eDeviceType_Media
eDeviceType_Plugin
eDeviceType_Script
eDeviceType_Security
eDeviceType_Thermostat

eDeviceType_Thermostat

```
Public Enum eDeviceType_Thermostat
      Operating_State = 1
      Temperature = 2
      Mode_Set = 3
      Fan_Mode_Set = 4
      Fan_Status = 5
      Setpoint = 6
      RunTime = 7
      Hold_Mode = 8
      Operating_Mode = 9
      Additional_Temperature = 10
      Setback = 11
      Filter_Remind = 12
      Root = 99                  ' Indicates a root device of a root/child grouping.
   End Enum
```

See Also

eDeviceType_GenericRoot
eDeviceType_Media
eDeviceType_Plugin
eDeviceType_Script
eDeviceType_Security
eDeviceType_SourceSwitch

Device_Type_Description (Read Only)

The Device_Type_Description read-only property is a string denoting the type of the device:

```
Public ReadOnly Property Device_Type_Description As String
```

The Device_API also determines what is returned by this property, as the value of the Device_Type is influenced by the API that the device subscribes to; if the API is the Thermostat API, then the device types are expected to be one of the eDeviceType_Thermostat enum values.

Example:

```
hs.WriteLog("Info", hs.DeviceName(dv.Ref) & " has a Device Type of " &
hs.DeviceType_Get(hs).Device_Type_Description)
```

See Also

- Device_API
- Device_API_Description (Read Only)
- Device_Type
- Device_SubType
- Device_SubType_Description

Device_SubType

The Device_SubType property is an integer denoting the device sub-type, if any, that this device is a part of:

```
Public Property Device_SubType As Integer
```

The Device_API should be set appropriately if the device is a part of an API, set to Plug_In if it is not a part of an API but is owned by a Plug-In, or set to No_API if it is not owned by a plug-in and is not a part of an API. If the Device_API is set to an API type such as Thermostat or Music, then the Device_Type should be set to one of the API types for those APIs. (See DeviceTypeInfo Enums) and this property can be used to denote the device type further.

NOTE: When the API is Thermostat, and the Device_Type is Setpoint, it is required that the Device_SubType be set to indicate which setpoint the device is representing, as found in the enum eDeviceSubType_Setpoint

Example:
(This sets the device to the Thermostat API type "Setpoint" for Cooling)

```
Dim DT As New DeviceAPI.DeviceTypeInfo
DT.Device_API = DeviceAPI.DeviceTypeInfo.eDeviceAPI.Thermostat
DT.Device_Type = DeviceAPI.DeviceTypeInfo.eDeviceType_Thermostat.Setpoint
DT.Device_SubType = DeviceAPI.DeviceTypeInfo.eDeviceSubType_Setpoint.Cooling_1
DT.Device_SubType_Description = "Cool Setpoint"
dv.DeviceType_Set(hs) = DT
hs.SaveEventsDevices
```

Example:
(This sets the device to a plug-in custom type.)

```
Dim DT As New DeviceAPI.DeviceTypeInfo
DT.Device_API = DeviceAPI.DeviceTypeInfo.eDeviceAPI.Plug_In
DT.Device_Type = CInt(PlugDeviceType_X)
DT.Device_SubType = 4
dv.DeviceType_Set(hs) = DT
hs.SaveEventsDevices
```

See Also

Device_API
Device_API_Description (Read Only)
Device_Type
Device_Type_Description (Read Only)
Device_SubType_Description

eDeviceSubType_SecurityArea

When the Device_Type is set to Security, and the security panel uses partitions/areas, the Device_SubType should be used to indicate the area number that the device belongs to IN ADDITION to there being a separate root/child device hierarchy per area/partition. When areas/partitions are NOT used, the Device_SubType can be any integer value, but to avoid misinterpretation, it is suggested that values below 20 not be used.

```
<Serializable()> _
Public Enum eDeviceSubType_SecurityArea
   Invalid = 0
   PRIMARY = 1
   Area_Partition_2 = 2
   Area_Partition_3 = 3
   Area_Partition_4 = 4
   Area_Partition_5 = 5
   Area_Partition_6 = 6
   Area_Partition_7 = 7
   Area_Partition_8 = 8
   Area_Partition_9 = 9
End Enum
```

See Also

eDeviceSubType_Setpoint

eDeviceSubType_Setpoint

When the Device_Type is set to a Thermostat API Type of Setpoint, the Device_SubType should be set to one of the enum values from this list.

```
<Serializable()> _
Public Enum eDeviceSubType_Setpoint
   Invalid = 0
   Heating_1 = 1
   Cooling_1 = 2
   Furnace = 7
   Dry_Air = 8
   Moist_Air = 9
   Auto_Changeover = 10
   Energy_Save_Heat = 11
   Energy_Save_Cool = 12
   Away_Heating = 13
End Enum
```

See Also

eDeviceSubType_SecurityArea

Device_SubType_Description

The Device_SubType_Description property is a string denoting the device sub-type, if any, that this device is a part of. The string is not used and is only for reference/description to the user.

```
Public Property Device_SubType_Description As String
```

This property is a description to go with the Device_SubType property to provide a meaningful description to the user of the device subtype.

Example:
(This sets the device to the thermostat API type "Setpoint")

```
Dim DT As New DeviceAPI.DeviceTypeInfo
DT.Device_API = DeviceAPI.DeviceTypeInfo.eDeviceAPI.Thermostat
DT.Device_Type = DeviceAPI.DeviceTypeInfo.eDeviceType_Thermostat.Setpoint
DT.Device_SubType = CInt(SPType)
DT.Device_SubType_Description = "Cool Setpoint"
dv.DeviceType_Set(hs) = DT
hs.SaveEventsDevices
```

Example:
(This sets the device to a plug-in custom type.)

```
Dim DT As New DeviceAPI.DeviceTypeInfo
DT.Device_API = DeviceAPI.DeviceTypeInfo.eDeviceAPI.Plug_In
DT.Device_Type = CInt(PlugDeviceType_X)
DT.Device_SubType = 4
DT.Device_SubType_Description = "Bazinga!"
dv.DeviceType_Set(hs) = DT
hs.SaveEventsDevices
```

See Also

- Device_API
- Device_API_Description (Read Only)
- Device_Type
- Device_Type_Description (Read Only)
- Device_SubType

Device_Type_String

Device_Type_String is a simple string description of the device type and has no actual bearing on the device as seen by HomeSeer or other plug-ins - only the DeviceType is used by HomeSeer and other plug-ins.

Example (Read):

```
If dv IsNot Nothing Then
 Log("My device has a device type of: " & dv.Device_Type_String(hs), LogType.Info)
End If
```

Example (Write):

```
If dv IsNot Nothing Then
 dv.Device_Type_String(hs) = "Joe Bazooka Bubble Gum"
End If
```

See Also

dvMISC
eRelationship
DeviceScriptChange
Device Value Status Pairs
Device Value Graphic Pairs
Device Type

Device Exists, Reference, Address and/or Code

In This Section

DeviceExistsRef
DeviceExistsAddress
DeviceExistsAddressFull
DeviceExistsCode
GetDeviceRef
GetDeviceRefByName
GetDeviceParentRefByRef
GetDeviceCode

See Also

The Device Class
Creating, Deleting, or Accessing Devices
Device Value, String, or Last Change
Device Energy Management
Device Control API (CAPI)

DeviceExistsRef

Purpose

This function indicates if the device does or does not exist.

Parameters

Parameter: **device**
Type: **integer**
Description: This is the device reference ID number.

Returns

Return value: **status**
Type: **Boolean**
Description: Returns `False` if the device does not exist.

See Also

DeviceExistsAddress
DeviceExistsAddressFull
DeviceExistsCode
GetDeviceRef
GetDeviceRefByName
GetDeviceParentRefByRef
GetDeviceCode

DeviceExistsAddress

Purpose

This function indicates if the device does or does not exist using its Address property (See also DeviceExistsCode).

Parameters

Parameter: **Address**
Type: **string**
Description: This is the device address, such as "Unit1", "0F47ED78-2", or "U2-I45-K2.2"

Parameter: **CaseSensitive**
Type: **Boolean**
Description: When True, the address must match exactly. When false, the address match is case insensitive such that apple2=APPLE2

Returns

Return value: **status**
Type: **long (.NET Integer)**
Description: Returns `-1` if the device does not exist, otherwise it returns the device reference ID number of the device. The reference number can then be used with the GetDeviceByRef function.

Note

The address field can contain any string of characters. The format and value is determined by a plug-in in the event that the device is owned by a plug-in.

When retrieved, the Address property includes the Code property, separated by a dash (-) if the Code property is set. For example, if the Address was set to "Unit1" and the code field is not used, then retrieving the Address field will result in "Unit1". If the Address was set to "Unit1" and the code was set to "Y55", then retrieving the Address field will result in "Unit1-Y55".

When this function is used, the Code field is NOT combined with the Address field. (See DeviceExistsAddressFull to find a device using its full address-code value.)

See Also

DeviceExistsRef
DeviceExistsAddressFull
DeviceExistsCode
GetDeviceRef
GetDeviceRefByName
GetDeviceParentRefByRef
GetDeviceCode

DeviceExistsAddressFull

Purpose

This function indicates if the device does or does not exist using its full Address-Code value (See also DeviceExistsAddress, DeviceExistsCode).

Parameters

Parameter: **Address**
Type: **string**
Description: This is the device address with code, such as "Unit1-R66", "0F47ED78-2", or "U2-I45-K2.2-Y55"

Parameter: **CaseSensitive**
Type: **Boolean**
Description: When True, the address-code must match exactly. When false, the address-code match is case insensitive such that apple2-Y65=APPLE2-y65

Returns

Return value: **status**
Type: **Integer**
Description: Returns -1 if the device does not exist, otherwise it returns the device reference ID number of the device. The reference number can then be used with the GetDeviceByRef function.

Note

The address field can contain any string of characters. The format and value is determined by a plug-in in the event that the device is owned by a plug-in.

When retrieved, the Address property includes the Code property, separated by a dash (-) if the Code property is set. For example, if the Address was set to "Unit1" and the code field is not used, then retrieving the Address field will result in "Unit1". If the Address was set to "Unit1" and the code was set to "Y55", then retrieving the Address field will result in "Unit1-Y55".

When this function is used, the Code field is combined with the Address field. (See DeviceExistsAddressl to find a device using its address value only.)

See Also

DeviceExistsRef
DeviceExistsAddress
DeviceExistsCode
GetDeviceRef
GetDeviceRefByName
GetDeviceParentRefByRef
GetDeviceCode

DeviceExistsCode

Purpose

This function indicates if the device does or does not exist using its Code property which is in the letter code and unit code format.

Parameters

Parameter: **Code**
Type: **String**
Description: This is the house/letter code and unit code of the device, such as "A1" or "q17".

Returns

Return value: **status**
Type: **Integer**
Description: Returns -1 if the device does not exist, otherwise it returns the device reference ID number of the device. The reference number can then be used with the GetDeviceByRef function.

See Also

DeviceExistsRef
DeviceExistsAddress
DeviceExistsAddressFull
GetDeviceRef
GetDeviceRefByName
GetDeviceParentRefByRef
GetDeviceCode

GetDeviceRef

Purpose

This function returns the device reference for a device. The device reference is different than an index to a device. The device reference is only needed for other procedures which explicitly require the device reference.

- This will only return the reference to the first device matching the address provided. The address is the "Address" property of a device, not the code property. The Address is normally set by a plugin and can be used to find a device.

Parameters

Parameter: **sAddress**
Type: **string**
Description: This is the actual device code, such as "12345-A1", or "IOPOINT1"

Returns

Return value: **reference**
Type: **Integer**
Description: This is a numerical device reference.

See Also

DeviceExistsRef
DeviceExistsAddress
DeviceExistsAddressFull
DeviceExistsCode
GetDeviceRefByName
GetDeviceParentRefByRef
GetDeviceCode

GetDeviceRefByName

Purpose

This function returns the device reference for a device. The device reference is different than an index to a device. The device reference is only needed for other procedures which explicitly require the device reference.

- This will only return the reference to the first device matching the name provided.

Parameters

Parameter: **sName**
Type: **string**
Description: This is the device name including the location, such as "Family Room Lamp".

Returns

Return value: **reference**
Type: **Integer**
Description: This is a numerical device reference.

Example

```
Sub Main()

    Dim dvRef
    Dim dv
```

```
dvRef = hs.GetDeviceRefByName("Family Room Light")
if dvRef > 0 then
                                Set dv = hs.GetDeviceByRef(dvRef)
else
   hs.WriteLog "Error","Could not find the reference for the device specified."
   exit Sub
end if
hs.WriteLog "Info","The address for the device is " & dv.hc & dv.dc

        End Sub
```

See Also

DeviceExistsRef
DeviceExistsAddress
DeviceExistsAddressFull
DeviceExistsCode
GetDeviceRef
GetDeviceParentRefByRef
GetDeviceCode

GetDeviceParentRefByRef

Purpose

This function returns a reference to a given device's parent device. If the device does not exist, then 0 will be returned. If the device reference number provided does not belong to a device, or if the device it references is not associated with a parent device, then 0 will be returned. See The Device Class for more information on associating devices.

Parameters

Parameter: **dvRef**
Type: **Integer**
Description: This is the reference ID of a device.

Returns

Return value: **dvRef**
Type: **Integer**
Description: Returns a reference to the given device's parent device.

See Also

DeviceExistsRef
DeviceExistsAddress
DeviceExistsAddressFull
DeviceExistsCode
GetDeviceRef
GetDeviceRefByName
GetDeviceCode

GetDeviceCode

Purpose

Returns the device code for the given named device. This function can be used with the IsOff and IsOn functions as well as other functions that require an actual device code.

Parameters

Parameter: **name**
Type: **string**
Description: This is the name of the device including its location, such as "den table lamp".

Returns

Return value: **Device Address and Code (if present)**
Type: **string**
Description: This is the address and code field of the device.

Example

```
dim code

code = hs.GetDeviceCode("den table lamp")

msgbox "The address is: " & code
```

See Also

DeviceExistsRef
DeviceExistsAddress
DeviceExistsAddressFull
DeviceExistsCode
GetDeviceRef
GetDeviceRefByName
GetDeviceParentRefByRef

Creating, Deleting, or Accessing Devices

In This Section

NewDeviceRef
NewDeviceEx
GetDeviceEnumerator
GetDeviceByRef
DeleteDevice
DeviceCount
DeviceButtonAdd

The application interface procedures in this section deal with creating a device, deleting a device, or getting a reference to a device so that a script or plug-in can work with it.

See Also

The Device Class
Device Exists, Reference, Address and/or Code
Device Value, String, or Last Change
Device Energy Management
Device Control API (CAPI)

NewDeviceRef

Purpose

This function creates a new device and gives it the specified name. The new device has the house code and unit code set to "A1", and all other attributes of the device are cleared. The device has no module type and has no location. Unlike NewDevice, this returns the unique device reference ID instead of an index, which makes this procedure more reliable during times of devices being added and removed by other scripts and plug-ins.

Parameters

Parameter: **name**
Type: **string**
Description: This is the name of the new device.

Returns

Return value: **device reference**
Type: **long**
Description: This is the device reference ID of the device that may be used in subsequent calls to the GetDeviceByRef function.

See Also

NewDeviceEx
GetDeviceEnumerator
GetDeviceByRef
DeleteDevice
DeviceCount
DeviceButtonAdd

NewDeviceEx

Purpose

This function creates a new device and sets its name property to the specified name. The house code and unit code of the device are set to A1. The returned object is a reference to the new device object. See the DeviceClass for a list of the properties available.

Parameters

Parameter: **name**
Type: **string**
Description: This is the name of the new device.

Returns

Return value: **reference to DeviceClass**
Type: **object**
Description: This returns a reference to the actual `DeviceClass` object. This allows direct access to all the properties of a device.

Example

```
sub main()

     dim dv

     set dv = hs.NewDeviceEx("my device")
     dv.location = "living room"

end sub
```

See Also

GetDeviceEnumerator

Purpose

This object can be used to iterate through all of the devices in HomeSeer, allowing you to work with the DeviceClass directly and make any changes or gather information that you need about events.

A `DeviceClass` has a number of properties that holds information about a device. You can access these properties to get and set this information.

Parameters

None.

Methods

Method: **GetNext**
Return value: **DeviceClass**
Type: **object**

Method: **Restart**
Return value: **none**
Type: **n/a**

Properties

Property: **Finished**
Type: **Boolean**
Description: TRUE when the enumerator reaches the last device.

Property: **CountChanged**
Type: **Boolean**
Description: TRUE when the count of devices changes during enumeration.

Example

The following script shows how to reiterate though all devices, get and display the device name.

```
Sub Main
    dim en
    dim dv

    set en = hs.GetDeviceEnumerator
    if IsObject(en) then
    else
        hs.WriteLog "Enumerator","The device enumerator is invalid."
    end if


    Do while not en.Finished
        if en.CountChanged then
            hs.WriteLog "Enumeration","-------- The device count has changed ----------"
        end if
        set dv = en.GetNext
        if not dv is nothing then
            hs.WriteLog "Enumeration","Got device " & dv.location & " " & dv.name
        end if
    Loop


End Sub
```

See Also

NewDeviceRef
NewDeviceEx
GetDeviceByRef
DeleteDevice
DeviceCount
DeviceButtonAdd

GetDeviceByRef

Purpose

This function returns a reference to the given device object. If the device does not exist, then an empty reference is returned.

Parameters

Parameter: **dvRef**
Type: **Integer**
Description: This is the reference ID of a device.

Returns

Return value: **device**
Type: **object as DeviceClass**
Description: Returns a reference to the given device object.

Example

```
Sub Main()


      Dim dvRef
      Dim dv

      dvRef = hs.GetDeviceRefByName("Family Room Light")
      if dvRef > 0 then
                           Set dv = hs.GetDeviceByRef(dvRef)
      else
          hs.WriteLog "Error","Could not find the reference for the device specified."
          exit Sub
      end if
      hs.WriteLog "Info","The address for the device is " & dv.hc & dv.dc

End Sub
```

See Also

NewDeviceRef
NewDeviceEx
GetDeviceEnumerator
DeleteDevice
DeviceCount
DeviceButtonAdd

DeleteDevice

Purpose

This function removes a device from HomeSeer. **Use this function with caution!**

Parameters

Parameter: **device_ref**
Type: **Integer**
Description: This is the device reference number.

Returns

Return value: **status**
Type: **boolean**
Description: Indicates the success or failure of the operation.

See Also

NewDeviceRef
NewDeviceEx
GetDeviceEnumerator
GetDeviceByRef
DeviceCount
DeviceButtonAdd

DeviceCount

Purpose

This function returns the total number of devices currently configured in the system.

Parameters

None.

Returns

Return value: **count of devices**
Type: **integer**

Example

```
dim count

count = hs.DeviceCount
```

See Also

NewDeviceRef
NewDeviceEx
GetDeviceEnumerator
GetDeviceByRef
DeleteDevice
DeviceButtonAdd

DeviceButtonAdd

This function has been deprecated. The enhanced functionality that this command used to provide can now be found by looking at the information in the Device Class ScriptName and ScriptFunc properties, the Device_Type/Device_API and Device_Type/eDeviceType_Script .

See Also

NewDeviceRef
NewDeviceEx
GetDeviceEnumerator
GetDeviceByRef
DeleteDevice
DeviceCount

Device Value, String, or Last Change

In This Section

The procedures below are for working with the device's Value, String, or the date/time it was last changed.

See Also

The Device Class
Device Exists, Reference, Address and/or Code
Creating, Deleting, or Accessing Devices
Device Energy Management
Device Control API (CAPI)

DeviceValue

Purpose

Returns the value stored for this device. Device values are double integer values that are associated with a device. This is a general-purpose value that you can set and read.

Parameters

Parameter: **dvRef**
Type: **Integer**
Description: This is the device reference ID number.

Returns

Return value: **Value**
Type: **Integer**
Description: This is the value stored for the device, which is usually the dim level.
Note: This return is an INTEGER value and values are DOUBLE INTEGER, which means that decimal values are truncated. See DeviceValueEx for Double Integer returns.

See Also

DeviceValueEx
DeviceValueByName
DeviceValueByNameEx
SetDeviceValue
SetDeviceValueByRef
SetDeviceValueByName
DeviceString
DeviceStringByName
SetDeviceString
SetDeviceStringByName
DeviceTime
DeviceTimeByName
DeviceDateTime
SetDeviceLastChange
DeviceLastChange
DeviceLastChangeRef
On - Off

DeviceValueEx

Purpose

Returns the value stored for this device. Device values are a double integer value that is associated with a device. This is a general-purpose value that you can set and read.

Parameters

Parameter: **dvRef**
Type: **Integer**
Description: This is the device reference ID number.

Returns

Return value: **Value**
Type: **Double Integer**
Description: This is the value stored for the device, which is usually the dim level.

See Also

DeviceValue
DeviceValueByName
DeviceValueByNameEx
SetDeviceValue
SetDeviceValueByRef
SetDeviceValueByName
DeviceString
DeviceStringByName
SetDeviceString
SetDeviceStringByName
DeviceTime
DeviceTimeByName
DeviceDateTime
SetDeviceLastChange
DeviceLastChange
DeviceLastChangeRef
On - Off

DeviceValueByName

Purpose

This function is the same as the DeviceValue function, except that you pass this function the *text name* of the device.

Parameters

Parameter: **device_name**
Type: **string**
Description: This is the name of the device and must contain both the location and name. If the device was named *lamp* and its location was *living room*, then the `device_name` parameter would be *living room lamp*.

Returns

Return value: **device value**
Type: **Integer**
Description: This returns the value associated with the device.
Note: This return is an INTEGER value and values are DOUBLE INTEGER, which means that decimal values are truncated. See DeviceValueByNameEx for Double Integer returns.

See Also

DeviceValue
DeviceValueEx
DeviceValueByNameEx
SetDeviceValue
SetDeviceValueByRef
SetDeviceValueByName
DeviceString
DeviceStringByName
SetDeviceString
SetDeviceStringByName
DeviceTime
DeviceTimeByName
DeviceDateTime
SetDeviceLastChange
DeviceLastChange
DeviceLastChangeRef
On - Off

DeviceValueByNameEx

Purpose

This function is the same as the DeviceValueEx function, except that you pass this function the *text name* of the device.

Parameters

Parameter: **device_name**
Type: **string**
Description: This is the name of the device and must contain both the location and name. If the device was named *lamp* and its location was *living room*, then the `device_name` parameter would be *living room lamp*.

Returns

Return value: **device value**
Type: **Double Integer**
Description: This returns the value associated with the device.

See Also

DeviceValue
DeviceValueEx
DeviceValueByName
SetDeviceValue
SetDeviceValueByRef
SetDeviceValueByName
DeviceString
DeviceStringByName
SetDeviceString
SetDeviceStringByName
DeviceTime
DeviceTimeByName
DeviceDateTime
SetDeviceLastChange
DeviceLastChange
DeviceLastChangeRef
On - Off

SetDeviceValue

Purpose

This function sets a value that is associated with this device. Values are used to hold the dim level of a device. You can also use them as user variables in your scripts. Note that HomeSeer will overwrite this value if a command was received for this device. If you are going to use this as storage for your own information, pick a device that does not exist in your home. You can also use virtual devices (devices in the range "q -> z" or unit codes between 17 and 64).

Parameters

Parameter: **device**
Type: **string**
Description: This is the device code, such as "A1".

Parameter: **value**
Type: **double integer**
Description: This is a numeric value, such as "50".

Returns

None.

Example

```
sub main()

      ' set the dim value of device B2 to 60.54
```

```
      hs.SetDeviceValue("B2", 60.54)
end sub
```

See Also

DeviceValue
DeviceValueEx
DeviceValueByName
DeviceValueByNameEx
SetDeviceValueByRef
SetDeviceValueByName
DeviceString
DeviceStringByName
SetDeviceString
SetDeviceStringByName
DeviceTime
DeviceTimeByName
DeviceDateTime
SetDeviceLastChange
DeviceLastChange
DeviceLastChangeRef
On - Off

SetDeviceValueByRef

Purpose

This function sets a value that is associated with this device. Values are used to hold the dim level of a device. You can also use them as user variables in your scripts. Note that HomeSeer will overwrite this value if a command was received for this device. If you are going to use this as storage for your own information, pick a device that does not exist in your home.

Parameters

Parameter: **dvRef**
Type: **Integer**
Description: This is the device reference ID number.

Parameter: **value**
Type: **double integer**
Description: This is a numeric value, such as "50".

Parameter: **trigger**
Type: **Boolean**
Description: When set to FALSE, the value will be changed without triggering events that are set to trigger when the device changes. Set this to True normally so that events can trigger when the device's value is updated.

Returns

None.

Example

```
Sub Main(ByVal Parms As Object)
      ' set the value of device whose reference ID is 1234 to 60.54
      hs.SetDeviceValueByRef(1234, 60.54, True)
end sub
```

See Also

DeviceValue
DeviceValueEx
DeviceValueByName
DeviceValueByNameEx
SetDeviceValue
SetDeviceValueByName
DeviceString
DeviceStringByName
SetDeviceString
SetDeviceStringByName
DeviceTime
DeviceTimeByName
DeviceDateTime
SetDeviceLastChange
DeviceLastChange
DeviceLastChangeRef
On - Off

SetDeviceValueByName

Purpose

This function is the same as SetDeviceValue except you pass this function the actual text name of the device.

Parameters

Parameter: **device_name**
Type: **string**
Description: This is the name of the device and must contain both the location and name. If the device was named *lamp* and its location was *living room*, the `device_name` parameter would be *living room lamp*.

Parameter: **value**
Type: **double integer**

Returns

None.

See Also

DeviceValue
DeviceValueEx
DeviceValueByName
DeviceValueByNameEx
SetDeviceValue
SetDeviceValueByRef
DeviceString
DeviceStringByName
SetDeviceString
SetDeviceStringByName
DeviceTime
DeviceTimeByName
DeviceDateTime
SetDeviceLastChange
DeviceLastChange
DeviceLastChangeRef
On - Off

DeviceString

Purpose

Returns the character string set for a device. See SetDeviceString.

Parameters

None.

Returns

Return value: **dvRef**
Type: **Integer**
Description: This is the device reference ID number for the device.

Example

```
sub main()

     dim s

     s=hs.DeviceString(5678)
     msgbox s

end sub
```

See Also

DeviceValue
DeviceValueEx
DeviceValueByName
DeviceValueByNameEx
SetDeviceValue
SetDeviceValueByRef
SetDeviceValueByName
DeviceStringByName
SetDeviceString
SetDeviceStringByName
DeviceTime
DeviceTimeByName
DeviceDateTime
SetDeviceLastChange
DeviceLastChange
DeviceLastChangeRef
On - Off

DeviceStringByName

Purpose

Returns the character string set for a device. See SetDeviceString.

Parameters

Parameter: **name**
Type: **string**
Description: This is the name of the device. The name includes its location, such as *den table lamp*.

Returns

None.

See Also

SetDeviceString

Purpose

This function sets a string as the device status. The string "message" is displayed in the Status screen. This appears on the web page and the local device list. This can be used to display the status of special devices like thermostats and weather stations. Note that this does not affect the actual status/value for the device, which can be accessed by DeviceValue.

The text string can also contain HTML code, so you can add affects to the status like changing its color or making it scroll. See the example below to create some status using the *marquee* and *blink* HTML tags. Note the *marquee* tag is only supported in Internet Explorer and the *blink* tag is only supported in Netscape.

Parameters

Parameter: **dvRef**
Type: **Integer**
Description: This is the device reference ID number.

Parameter: **message**
Type: **string**
Description: This is the status string for the device, such as "72 degrees".

Parameter: **reset**
Type: **boolean**
Description: If this is set to TRUE, the device change date/time will be updated (normally a string change will not update the device last change date/time).

Returns

None.

Example

```
sub main()

      hs.SetDeviceString(5678, Motion Detected", True)

      ' add some HTML to the text to create a scrolling status

      hs.SetDeviceString(5678, "<MARQUEE><blink><b>Motion Detected</MARQUEE> </b></blink>", True)

end sub
```

See Also

DeviceValue
DeviceValueEx
DeviceValueByName
DeviceValueByNameEx
SetDeviceValue
SetDeviceValueByRef
SetDeviceValueByName
DeviceString
DeviceStringByName
SetDeviceStringByName
DeviceTime
DeviceTimeByName
DeviceDateTime
SetDeviceLastChange
DeviceLastChange
DeviceLastChangeRef
On - Off

SetDeviceStringByName

Purpose

This function sets a string as the device status using the actual name of the device combined with its location. See SetDeviceString.

Parameters

Parameter: **name**
Type: **string**
Description: This is the name of the device including it location, such as *den table lamp*. Note the name is not case-sensitive.

Parameter: **message**
Type: **string**
Description: This is the status string for the device, such as *72 degrees*.

Parameter: **reset**
Type: **boolean**
Description: If this is set to TRUE, the device change date/time will be updated (normally a string change will not update the device last change date/time).

Returns

None.

See Also

DeviceValue
DeviceValueEx
DeviceValueByName
DeviceValueByNameEx
SetDeviceValue
SetDeviceValueByRef
SetDeviceValueByName
DeviceString
DeviceStringByName
SetDeviceString
DeviceTime
DeviceTimeByName
DeviceDateTime
SetDeviceLastChange
DeviceLastChange
DeviceLastChangeRef
On - Off

DeviceTime

Purpose

Returns the time in minutes since the device status last changed. This can be used to see how long a device has been ON or OFF.

Parameters

Parameter: **dvRef**
Type: **Integer**
Description: This is the device reference ID number.

Returns

Return value: **time**
Type: **integer**
Description: This is the amount of time in minutes since last device change.

See Also

DeviceValue
DeviceValueEx
DeviceValueByName
DeviceValueByNameEx
SetDeviceValue
SetDeviceValueByRef
SetDeviceValueByName
DeviceString
DeviceStringByName
SetDeviceString
SetDeviceStringByName
DeviceTimeByName
DeviceDateTime
SetDeviceLastChange
DeviceLastChange
DeviceLastChangeRef
On - Off

DeviceTimeByName

Purpose

Returns the time in minutes since the device status last changed. This can be used to see how long a device has been ON or OFF.

Parameters

Parameter: **device name**
Type: **string**
Description: This is the name of the device and it must contain both the location and name. If the device was named *lamp* and its location was *living room*, then the `device name` parameter would be *living room lamp*.

Returns

Return value: **time**
Type: **integer**
Description: This is the amount of time in minutes since last device status change.

See Also

DeviceValue
DeviceValueEx
DeviceValueByName
DeviceValueByNameEx
SetDeviceValue
SetDeviceValueByRef
SetDeviceValueByName
DeviceString
DeviceStringByName
SetDeviceString
SetDeviceStringByName
DeviceTime
DeviceDateTime
SetDeviceLastChange
DeviceLastChange
DeviceLastChangeRef
On - Off

DeviceDateTime

Purpose

Returns the time as a date/time object, of when the device last changed value or string.

Parameters

Parameter: **dvRef**
Type: **Integer**
Description: This is the device reference ID number.

Returns

Return value: **time**
Type: **Date**
Description: This is the date and time of the last change to the device.

See Also

DeviceValue
DeviceValueEx
DeviceValueByName
DeviceValueByNameEx
SetDeviceValue
SetDeviceValueByRef
SetDeviceValueByName
DeviceString
DeviceStringByName
SetDeviceString
SetDeviceStringByName
DeviceTime
DeviceTimeByName
SetDeviceLastChange
DeviceLastChange
DeviceLastChangeRef
On - Off

SetDeviceLastChange

Purpose

This function sets the last change time of a device.

Parameters

Parameter: **dvRef**
Type: **Integer**
Description: This is the device reference ID number.

Parameter: **date-time**
Type: **date**
Description: This is the date and time to set the last change to.

Returns

Return value: **none**

Example

```
hs.SetDeviceLastChange(5678, Now)
hs.SetDeviceLastChange(5678, Convert.ToDateTime("1/1/13 4:00 PM"))
```

See Also

DeviceValue
DeviceValueEx
DeviceValueByName
DeviceValueByNameEx
SetDeviceValue
SetDeviceValueByRef
SetDeviceValueByName
DeviceString
DeviceStringByName
SetDeviceString
SetDeviceStringByName
DeviceTime
DeviceTimeByName
DeviceDateTime
DeviceLastChange
DeviceLastChangeRef
On - Off

DeviceLastChange

Purpose

This function returns the date and time the device last changed its value or character string.

Parameters

Parameter: **device**
Type: **string**
Description: This is the house code and unit code or device code of the device, such as "A1" or "q17".

Returns

Return value: **device time**
Type: **date**
Description: This is the date and time the device last changed.

Example

```
sub main()
```

```
    Dim last_change As Date

    ' get the last change time for the device name "living room lamp"
    last_change = hs.DeviceLastChange("living room lamp")

end sub
```

See Also

DeviceValue
DeviceValueEx
DeviceValueByName
DeviceValueByNameEx
SetDeviceValue
SetDeviceValueByRef
SetDeviceValueByName
DeviceString
DeviceStringByName
SetDeviceString
SetDeviceStringByName
DeviceTime
DeviceTimeByName
DeviceDateTime
SetDeviceLastChange
DeviceLastChangeRef
On - Off

DeviceLastChangeRef

Purpose

This function returns the date and time the device last changed status.

Parameters

Parameter: **dvRef**
Type: **Integer**
Description: This is the device reference ID number.

Returns

Return value: **device time**
Type: **date**
Description: This is the date and time the device last changed.

Example

```
sub main()

    Dim last_change As Date

    ' get the last change time for the device whose reference ID number is 5678
    last_change = hs.DeviceLastChangeRef(5678)

end sub
```

See Also

DeviceValue
DeviceValueEx
DeviceValueByName
DeviceValueByNameEx
SetDeviceValue
SetDeviceValueByRef
SetDeviceValueByName
DeviceString
DeviceStringByName
SetDeviceString
SetDeviceStringByName
DeviceTime
DeviceTimeByName
DeviceDateTime
SetDeviceLastChange
DeviceLastChange
On - Off

On - Off

These functions are largely deprecated due to the nature of devices not always being the same value or state for On and Off - for example, an X-10 device may have an "On" value of 100, but a Z-Wave device is "On" at values 99 (dimmable device) or 255 (non-dimmable).

For backward compatibility, these functions return True for their query if the value is 0 (for Off), or in the range 1 to 100 or 255 (for On).

See Also

DeviceValue
DeviceValueEx
DeviceValueByName
DeviceValueByNameEx
SetDeviceValue
SetDeviceValueByRef
SetDeviceValueByName
DeviceString
DeviceStringByName
SetDeviceString
SetDeviceStringByName
DeviceTime
DeviceTimeByName
DeviceDateTime
SetDeviceLastChange
DeviceLastChange
DeviceLastChangeRef

IsOn

Purpose

This function checks the status of a device.

Parameters

Parameter: **device**
Type: **string**
Description: This is the house code and unit code or device code of the device, such as "A1" or "q17".

Returns

Return value: **status**
Type: **boolean**
Description: This returns TRUE if the device is on or dimmed or FALSE if the device is off.

Example

```
sub main()

     if hs.IsOn("A1") then
                              hs.speak "the light is on"
     end if

end sub
```

See Also

IsOnByName
IsOff
IsOffByName

IsOnByName

Purpose

This function checks the status of a device, using the device's name.

Parameters

Parameter: **device name**
Type: **string**
Description: This is the name of the device. The device name must include the device's location and its name, such as "den table lamp". The name is not case-sensitive.

Returns

Return value: **status**
Type: **boolean**
Description: This returns TRUE if a device is on or dimmed or FALSE if it's off.

See Also

IsOn
IsOff
IsOffByName

IsOff

Purpose

This function checks the status of a device.

Parameters

Parameter: **device**
Type: **string**
Description: This is the house code and unit code or device code of the device, such as "A1" or "q17".

Returns

Return value: **status**
Type: **boolean**
Description: This returns TRUE if the device is OFF and FALSE if it's not (on or dimmed).

See Also

IsOn
IsOnByName
IsOffByName

IsOffByName

Purpose

This function checks the status of a device, using the device's name.

Parameters

Parameter: **device name**
Type: **string**
Description: This is the name of the device. The device name must include the device's location and its name like "den table lamp". The name is not case-sensitive.

Returns

Return value: **status**
Type: **boolean**
Description: This returns TRUE if a device is off and FALSE if it's on or dimmed.

See Also

IsOn
IsOnByName
IsOff

Device Energy Management

The methods and objects in this section provide a method of storing energy information and calculating energy usage and cost on a per-device basis. When energy information is added, it is stored in the Energy database. "Calculators" can be added to a device's Energy object to track energy amounts and cost over periods of time. If you want to know the energy used and the cost for the past hour, so far today, yesterday, and last week, that is four calculators. For graphing or more intensive calculations, a method is also provided which allows energy data records to be returned given a starting date/time and a length of time.

See Also

Energy_AddData, Energy_AddDataArray

These methods are used to add one record of energy data to a device (_AddData) or several records (_AddDataArray). If the return value from this procedure is False, then HomeSeer was unable to accept the data and add it to the data queue to be added to the database. See the EnergyData Class for more information about the information stored in the object referenced in these procedures.

```
Function Energy_AddData(ByVal dvRef As Integer, ByVal Data As EnergyData) As Boolean
Function Energy_AddDataArray(ByVal dvRef As Integer, ByVal colData As EnergyData()) As Boolean
```

See Also

Energy_SetEnergyDevice
Energy_AddCalculator, Energy_AddCalculatorEvenDay
Energy_CalcCount
Energy_GetCalcByName, Energy_GetCalcByIndex
Energy_GetData, Energy_GetArchiveData
Energy_RemoveData

EnergyData Class

The EnergyData object holds a set of energy usage or consumption information, and the rate of that energy at the time it was consumed or produced.

```
Public Class EnergyData

    Public dvRef As Integer                  'Device reference ID number for the device this energy data is for.
    Public Direction As enumEnergyDirection = enumEnergyDirection.Consumed   'Indicates whether the energy was
    '                                                                        consumed (used) or produced (created).
    Public Amount As Double                  'Always measured in Watts
    Public Amount_Start As Date              'The start of the time period this measurement is for.
    Public Amount_End As Date                'The end of the time period this measurement is for.
    Public Rate As Single                    'Always measured in kWH
    Public UserCode As Integer               'For the user to indicate something about this reading.

End Class
```

Note: When initialized, the value of Direction must be provided:
Example:

```
Dim ED As New EnergyData(enumEnergyDirection.Consumed)
```

See Also

enumEnergyDevice

These enum values can be used in the EnergyData to indicate the type of device the energy data came from.

```
Public Enum enumEnergyDevice
    _Undefined_ = 0              ' Not defined
    Light_Small = 1              ' A small light
    Light_Large = 2              ' A large light or several lights
    Appliance = 10               ' Any appliance
    Appliance_Small = 11         ' A small appliance such as a toaster
    Appliance_Large = 12         ' A large appliance such as an oven
    Utility = 20                 ' A utility device
    Utility_Small = 21           ' A small utility device such as a water filter
    Utility_Large = 22           ' A large utility device such as a well pump
    Entertainment = 30           ' An entertainment device
    Entertainment_Small = 31     ' A small entertainment device such as a radio
    Entertainment_Large = 32     ' A large entertainment device such as a home theatre system
    HVAC = 40                    ' An HVAC device
    Electric_AC = 41             ' An Air Conditioning device
    Electric_Heat = 42           ' An electric heating device
    Panel = 51                   ' An electrical panel providing several branches of electrical service to the
home.
    Panel_A = 52                 '
    Panel_B = 53                 '
    Panel_C = 54                 '
    Panel_D = 55                 '
    Panel_E = 56                 '
    Panel_F = 57                 '
    Meter = 61                   ' An electric meter measuring usage for an unspecified or general purpose.
    Meter_Service = 62           ' An electric meter measuring usage for electrical service such as a house
service entrance.
    Meter_Device = 63            ' An electric meter measuring usage for a single device.
    Generator = 71               ' An electricity producing generator.
    Solar_Panel = 72             ' An electricity producing solar panel.
    Wind_Turbine = 73            ' An electricity producing wind turbine.
    Water_Turbine = 74           ' An electricity producing water (wave) turbine.
    Other = 99                   ' A device (consumer or producer) that does not fit any other device type.
End Enum
```

See Also

enumEnergyDirection

enumEnergyDirection

This enum is used in the EnergyData to indicate whether the energy information is for energy consumed or produced.

```
Public Enum enumEnergyDirection
    Consumed = 1
    Produced = 2
```

```
End Enum
```

See Also

enumEnergyDevice

Energy_SetEnergyDevice

This procedure is used to set the type of energy consumption or energy producing device for the reference ID dvRef in the energy database. This procedure is also used to create an initial energy object in the system if one does not exist. The energy object's device name, location, and location2 properties will also be updated whenever this procedure is called.

```
Public Function Energy_SetEnergyDevice(ByVal dvRef As Integer, _
                                       ByVal DeviceType As enumEnergyDevice) As Boolean
```

See Also

Energy_AddData, Energy_AddDataArray
Energy_AddCalculator, Energy_AddCalculatorEvenDay
Energy_CalcCount
Energy_GetCalcByName, Energy_GetCalcByIndex
Energy_GetData, Energy_GetArchiveData
Energy_RemoveData

enumEnergyDevice

These enum values can be used in the EnergyData to indicate the type of device the energy data came from.

```
Public Enum enumEnergyDevice

   _Undefined_ = 0              ' Not defined
   Light_Small = 1              ' A small light
   Light_Large = 2              ' A large light or several lights
   Appliance = 10               ' Any appliance
   Appliance_Small = 11         ' A small appliance such as a toaster
   Appliance_Large = 12         ' A large appliance such as an oven
   Utility = 20                 ' A utility device
   Utility_Small = 21           ' A small utility device such as a water filter
   Utility_Large = 22           ' A large utility device such as a well pump
   Entertainment = 30           ' An entertainment device
   Entertainment_Small = 31     ' A small entertainment device such as a radio
   Entertainment_Large = 32     ' A large entertainment device such as a home theatre system
   HVAC = 40                    ' An HVAC device
   Electric_AC = 41             ' An Air Conditioning device
   Electric_Heat = 42           ' An electric heating device
   Panel = 51                   ' An electrical panel providing several branches of electrical service to the
home.
   Panel_A = 52                 '
   Panel_B = 53                 '
   Panel_C = 54                 '
   Panel_D = 55                 '
   Panel_E = 56                 '
```

```
    Panel_F = 57                '
    Meter = 61                  ' An electric meter measuring usage for an unspecified or general purpose.
    Meter_Service = 62          ' An electric meter measuring usage for electrical service such as a house
service entrance.
    Meter_Device = 63           ' An electric meter measuring usage for a single device.
    Generator = 71              ' An electricity producing generator.
    Solar_Panel = 72            ' An electricity producing solar panel.
    Wind_Turbine = 73           ' An electricity producing wind turbine.
    Water_Turbine = 74          ' An electricity producing water (wave) turbine.
    Other = 99                  ' A device (consumer or producer) that does not fit any other device type.

End Enum
```

See Also

Energy_AddCalculator, Energy_AddCalculatorEvenDay

These functions are used to add an energy calculator to an energy object in HomeSeer. An energy calculator updates when energy data is added, and calculates the total energy consumed or produced for the time period, as well as the cost of that energy. The return value is a string that is empty if the procedure succeeded, and contains error information if it did not.

```
Function Energy_AddCalculator(ByVal dvRef As Integer, ByVal Name As String, ByVal Range As TimeSpan, ByVal StartBack As TimeSpan) As String
```

```
Function Energy_AddCalculatorEvenDay(ByVal dvRef As Integer, ByVal Name As String, ByVal Range As TimeSpan, ByVal StartBack As TimeSpan) As String
```

Calculators that are for days, for example the amount of energy used a week ago today, can use _AddCalculatorEvenDay and the calculation will automatically be truncated at even day boundaries.

Parameters

Parameter: **dvRef**
Type: **Integer**
Description: The unique device reference ID number.

Parameter: **Name**
Type: **String**
Description: This is the name of the calculator, which may be used to identify the calculation being done.

Parameter: **Range**
Type: **TimeSpan**
Description: This is the time period that you wish the calculation to be done over.

Parameter: **StartBack**
Type: **TimeSpan**
Description: This is the period of time, starting from "Now", to go back to and set as the start time for the calculation.

Returns

Return value: **Result**
Type: **String**
Description: This is the result of the operation - if it succeeded, it will be an empty string - if it failed, it will contain information about the error.

Example:

To create a calculator for the energy used in the past hour:

```
Result = hs.Energy_AddCalculator(1234, "Last Hour Used", New TimeSpan(1, 0, 0), New TimeSpan(0, 0, 0))
```

```
If Not String.IsNullOrEmpty(Result) Then
 hs.WriteLog("Error", "Calculator add failed, reason=" & Result)
End If
```

To create a calculator for the energy used in the past hour yesterday:

```
Result = hs.Energy_AddCalculator(1234, "Last Hour Used Yesterday", New TimeSpan(1, 0, 0), New TimeSpan(1, 0, 0, 0))
```

See Also

Energy_AddData, Energy_AddDataArray
Energy_SetEnergyDevice
Energy_CalcCount
Energy_GetCalcByName, Energy_GetCalcByIndex
Energy_GetData, Energy_GetArchiveData
Energy_RemoveData

Energy_CalcCount

This function will return the number of energy calculators currently attached to the energy object for the device referenced by dvRef. This can be used in conjunction with Energy_GetCalcByIndex to retrieve all of the calculator data for a device.

```
Function Energy_CalcCount(ByVal dvRef As Integer) As Integer
```

See Also

Energy_AddData, Energy_AddDataArray
Energy_SetEnergyDevice
Energy_AddCalculator, Energy_AddCalculatorEvenDay
Energy_GetCalcByName, Energy_GetCalcByIndex
Energy_GetData, Energy_GetArchiveData
Energy_RemoveData

Energy_GetCalcByName, Energy_GetCalcByIndex

These functions return energy calculation results for the device referenced by dvRef. The EnergyCalcData objects contain energy results as well as other parameters used when the calculator was created, but they are for reference only and changing those properties will NOT be reflected back to the real calculator object on the device.

See the EnergyCalcData Class object definition for more information about its members.

Use Energy_GetCalcByIndex after using Energy_CalcCount to iterate through each calculator on an energy object without having to know its name.

```
Function Energy_GetCalcByName(ByVal dvRef As Integer, ByVal Name As String) As EnergyCalcData
Function Energy_GetCalcByIndex(ByVal dvRef As Integer, ByVal Index As Integer) As EnergyCalcData
```

See Also

Energy_AddData, Energy_AddDataArray
Energy_SetEnergyDevice
Energy_AddCalculator, Energy_AddCalculatorEvenDay
Energy_CalcCount
Energy_GetData, Energy_GetArchiveData
Energy_RemoveData

EnergyCalcData Class

This class object is used as a return value from Energy_GetCalcByName or Energy_GetCalcByIndex. It contains the results of the most recent data calculation performed on the energy data added to the device.

```
Public Class EnergyCalcData
    Public Range As TimeSpan                        ' The amount of time to be included in the calculation
                                                    '      starting from the starting point.
    Public StartBack As TimeSpan                    ' The amount of time to be subtracted from NOW to
                                                    '      get our starting point.
    Public RoundDay As Boolean = False              ' Whether to round the time to an even day.
    Public Property Name As String                  ' The name of the energy calculator this data belongs to.
    Public ReadOnly Property Amount As Double       ' The amount of energy rounded to 3 decimal places.
    Public ReadOnly Property AmountPrecise As Double    ' The amount of energy without any rounding.
    Public ReadOnly Property Cost As Double         ' The cost of the energy calculated, rounded
                                                    '      to 2 decimal places.
    Public ReadOnly Property CostPrecise As Double  ' The cost of the energy calculated without any rounding.
End Class
```

See Also

Energy_GetData, Energy_GetArchiveData

For doing your own calculations or graphing, these functions allow you to get energy data for a device. Use _GetData to retrieve recent data, as far back as the oldest information used by one of the calculators, or _GetArchiveData to retrieve any range of data using a start and end date.

```
Function Energy_GetData(ByVal dvRef As Integer, _
                        ByVal dteStart As Date, ByVal dteEnd As Date)
        As Collections.Generic.List(Of EnergyData)

Function Energy_GetArchiveData(ByVal dvRef As Integer, _
                               ByVal dteStart As Date, ByVal dteEnd As Date)
        As Collections.Generic.List(Of EnergyData)
```

See Also

Energy_AddData, Energy_AddDataArray
Energy_SetEnergyDevice
Energy_AddCalculator, Energy_AddCalculatorEvenDay
Energy_CalcCount
Energy_GetCalcByName, Energy_GetCalcByIndex
Energy_RemoveData

EnergyData Class

The EnergyData object holds a set of energy usage or consumption information, and the rate of that energy at the time it was consumed or produced.

```
Public Class EnergyData

    Public dvRef As Integer                    'Device reference ID number for the device this energy data is for.
    Public Direction As enumEnergyDirection = enumEnergyDirection.Consumed   'Indicates whether the energy was
    '                                                                         consumed (used) or produced (created).
    Public Amount As Double                    'Always measured in Watts
    Public Amount_Start As Date                'The start of the time period this measurement is for.
    Public Amount_End As Date                  'The end of the time period this measurement is for.
    Public Rate As Single                      'Always measured in kWH
    Public UserCode As Integer                 'For the user to indicate something about this reading.

End Class
```

Note: When initialized, the value of Direction must be provided:
Example:

```
Dim ED As New EnergyData(enumEnergyDirection.Consumed)
```

See Also

enumEnergyDevice

These enum values can be used in the EnergyData to indicate the type of device the energy data came from.

```
Public Enum enumEnergyDevice

    _Undefined_ = 0              ' Not defined
    Light_Small = 1              ' A small light
    Light_Large = 2              ' A large light or several lights
    Appliance = 10               ' Any appliance
    Appliance_Small = 11         ' A small appliance such as a toaster
    Appliance_Large = 12         ' A large appliance such as an oven
    Utility = 20                 ' A utility device
    Utility_Small = 21           ' A small utility device such as a water filter
    Utility_Large = 22           ' A large utility device such as a well pump
    Entertainment = 30           ' An entertainment device
    Entertainment_Small = 31     ' A small entertainment device such as a radio
    Entertainment_Large = 32     ' A large entertainment device such as a home theatre system
    HVAC = 40                    ' An HVAC device
    Electric_AC = 41             ' An Air Conditioning device
    Electric_Heat = 42           ' An electric heating device
    Panel = 51                   ' An electrical panel providing several branches of electrical service to the home.
    Panel_A = 52                 '
    Panel_B = 53                 '
    Panel_C = 54                 '
    Panel_D = 55                 '
    Panel_E = 56                 '
    Panel_F = 57                 '
    Meter = 61                   ' An electric meter measuring usage for an unspecified or general purpose.
```

```
    Meter_Service = 62              ' An electric meter measuring usage for electrical service such as a house
service entrance.
    Meter_Device = 63               ' An electric meter measuring usage for a single device.
    Generator = 71                  ' An electricity producing generator.
    Solar_Panel = 72                ' An electricity producing solar panel.
    Wind_Turbine = 73               ' An electricity producing wind turbine.
    Water_Turbine = 74              ' An electricity producing water (wave) turbine.
    Other = 99                      ' A device (consumer or producer) that does not fit any other device type.
End Enum
```

See Also

enumEnergyDirection

enumEnergyDirection

enumEnergyDirection

See Also

enumEnergyDevice

Energy_RemoveData

This command will remove energy data for the device referenced by dvRef, from the date/time specified in dteStart and older.

```
Function Energy_RemoveData(ByVal dvRef As Integer, ByVal dteStart As Date) As Integer
```

Example:

To remove energy records from the system and the database which are more than a year old...

```
Dim Result As Integer
Result = hs.Energy_RemoveData(1234, Now.Subtract(New TimeSpan(365, 0, 0, 0)))
If Result < 0 Then
    hs.WriteLogEx("Error", "Error removing energy data from a year ago for " & DeviceName(1234),
COLOR_RED)
Else
    hs.WriteLog("Info", "Maintenance on energy data for " & DeviceName(1234) & _
                        " resulted in " & Result.ToString & " records being removed.")
End If
```

See Also

Energy_AddData, Energy_AddDataArray
Energy_SetEnergyDevice
Energy_AddCalculator, Energy_AddCalculatorEvenDay
Energy_CalcCount
Energy_GetCalcByName, Energy_GetCalcByIndex
Energy_GetData, Energy_GetArchiveData

Device Control API (CAPI)

The device control API is the sole method for scripts and plug-ins to control devices. The object of the device type class model is to provide a way for devices to be self-describing in how they are to be visually rendered, how they are to be controlled, and for as many common device types as we can create that they subscribe to a standard model that enables other applications to use them.

The device control API is a means for programmers creating alternative interfaces for HomeSeer to be able to obtain status information and control devices, regardless of whether the device was designed to use buttons, status, or values (or a combination) as the main control mechanism.

How It Works

The device control API works from the premise that HomeSeer always knows how to render a device's status and provide control options on the device status page. All devices will use status/values/graphics pairs to represent both control and status information. If the device has buttons associated with it, the buttons are displayed. If the device has a string value, the string is displayed under the status column, etc.... Using the code that HomeSeer uses, the control API returns a status object for a device, and a collection of control capabilities can be obtained. Codes in the control capability objects tell the user of the API what kind of a control method is used and the data/value that corresponds to it.

All of the procedures of the device control API use the unique device reference number (dv.ref) - it is best to have a good working knowledge of the HomeSeer scripting interface commands that work with the device reference IDs. In most cases, the device enumerator is your friend.

The pages herein describe the API procedures and objects.

A plug-in will access the control API through the HomeSeer scripting interface or HSApplicationAPI interface. This is accessed through the "hs" object that is obtained when a plug-in connects to HomeSeer.

See Also

The Device Class
Device Exists, Reference, Address and/or Code
Creating, Deleting, or Accessing Devices
Device Value, String, or Last Change
Device Energy Management

CAPIGetStatus

```
Public Function CAPIGetStatus(ByVal dvRef As Integer) As iCAPIStatus
```

The return value is a CAPIStatus object. See the iCAPIStatus subject for a description of its properties and members.

Example

```
Sub Main(ByVal Parm As Object)
    Dim enx As Scheduler.clsDeviceEnumeration
    Dim dv As Scheduler.Classes.DeviceClass
    Dim CS As Scheduler.CAPIStatus
    Dim s As String

    enx = hs.GetDeviceEnumerator()
    If enx Is Nothing Then
    Else
        hs.WriteLog("Test", "It is not an object.")
    End If

    Do While enx.Finished = False
        dv = enx.GetNext()
        If Not dv Is Nothing Then
            CS = hs.CAPIGetStatus(dv.ref)
            s = "Device " & dv.location & " " & dv.Name
            s &= ", Status=" & CS.Status & ", Image=" & CS.ImageFile
            hs.WriteLog("Enumeration", s)
        End If
    Loop

End Sub
```

See Also

CAPIGetControl, CAPIGetControlEx, CAPIGetSingleControl
CAPIControlHandler, CAPIControlsHandler

iCAPIStatus

iCAPIStatus is an interface class with the following definition, used with the CAPIStatus procedures:

```
Public Interface ICAPIStatus

    Property Status() As String
    Property StatusHTML() As String
    Property ImageFile() As String
    Property ClassName() As String
    Property Value As Double

End Interface
```

Property	Description
Status	This is the status text as displayed on the HomeSeer device utility page and other UI pages.
StatusHTML	If the device's status contains HTML, which is sometimes stripped away when the status is displayed, this property contains the status with HTML.
ImageFile	This is the path to a graphics file that corresponds with the device's current status value. It is the path to the same graphic as shown on the device utility page.
ClassName	This is the class name assigned to the cell that the device status is displayed within on the HomeSeer device utility page.
Value	This is the current value of the device which corresponds to the status information present in the other properties.

See Also

CAPIGetControl, CAPIGetControlEx, CAPIGetSingleControl

```
Public Function CAPIGetControl(ByVal dvRef As Integer) As CAPIControl()

Public Function CAPIGetControlEx(ByVal dvRef As Integer, _
                                 ByVal SingleRangeEntry As Boolean) As CAPIControl()

Public Function CAPIGetSingleControl(ByVal dvRef As Integer, _
                                     ByVal SingleRangeEntry As Boolean, _
                                     ByVal Label As String, _
                                     ByVal ExactCase As Boolean, _
                                     ByVal Contains As Boolean) As CAPIControl
```

Because a device has multiple control options, two of these functions return an array of CAPIControl objects, the third returns a specific control option.

See CAPIControl for more information about the object or array of objects returned by these functions.

See Also

CAPIGetStatus
CAPIControlHandler, CAPIControlsHandler

CAPIControl

The CAPIControl object holds information about a control state for a device. Generated from value/status, value/graphic pairs, and buttons, CAPIControl objects are used by other plug-ins to render control options for a device properly, and to invoke a control method on a device.

Here are the members of CAPIControl:

```
Public Class CAPIControl
   Inherits MarshalByRefObject
   Public Do_Update As Boolean
   Public SingleRangeEntry As Boolean
   Public Property CCIndex As Integer
   Public Property Range As clsValueRange
   Public ReadOnly Property Ref As Integer
   Public Property Label As String
   Public Property ControlType As Enums.CAPIControlType
   Public Property ControlLocation As Enums.CAPIControlLocation
   Public Property ControlLoc_Row As Integer
   Public Property ControlLoc_Column As Integer
   Public Property ControlValue As Double
   Public Property ControlString As String = ""
   Public Property ControlStringList() As String()
   Public Property ControlFlag As Boolean
End Class
```

The definition for each member is as follows:

Name	Description
Do_Update	The default value of this is True, set it to False to prevent HomeSeer from triggering events based upon this control action taking place.

SingleRangeEntry	This property tracks, and has no effect when SET, the value of SingleRangeEntry when this control option was generated. SingleRangeEntry is an option on CAPIGetControl which determines whether ranges are provided as a single range entry, or multiple single-value entries which are generated by HomeSeer based upon the minimum and maximum values of the range.
CCIndex	This is the index of the CAPIControl object. It is not guaranteed to remain constant against each unique control option and may change.
Range	This class object is null (Nothing) if the control object does not represent a range of values. If the control object represents a range of values, it will be populated with the contents of a clsValueRange object.
Ref	This is the unique device reference ID for which this control object is for. (Read Only)
Label	The label property contains the text "status" for this control value. For example, if this control object were generated from a button added to the device with the name "Play", then this property will be "Play". You may search through the CAPIControl object for a device to find the desired control object using the label field to match one of the control options displayed on the device utility page for a device.
ControlType	This indicates the type of control used to achieve the state described by the object, which also describes the way in which the control should be rendered in a user interface. There are two types of control, a value or a string. Values are Double Integer and are used by most of the control types, string control options do NOT change the device value and are used only with the TextList and TextBox_String ControlTypes. One additional special ControlType exists of Button_Custom, which calls a script when the control is handled. All other control types cause the device's value to be set to the corresponding value.
ControlLocation	This property allows access to the Row, Column and ColumnSpan values in one structure (CAPIControlLocation). The row and column values are populated with the positive row number and column number of where the control should be rendered in a grid. If the row/column is zero, the control should NOT be drawn on the UI (this may be used to provide a control that plug-ins and scripts can access but should not be shown on the user UI). If the CAPIControl object was created from device value/status pairs, then the row and column values in this structure were copied from the value/status pair which created this particular CAPIControl. The ColumnSpan may be used to better align and place the generated controls - for example, if you have 3 smaller buttons and then a large (wide) slider, place the buttons on row 1 and the slider on row 2, and specify the slider to be 3 columns wide, and that will result in the slider being rendered directly below the 3 buttons, rather than the default of the slider forcing one of the button columns to be extra wide.
ControlLoc_Row	This is the Row property of the ControlLocation accessible as an individual integer value.
ControlLoc_Column	This is the Column property of the ControlLocation accessible as an individual integer value.
ControlValue	When the ControlType is one of the types that indicate a value is used to achieve the desired control state, this property holds the value to be used.
ControlString	When the ControlType is one of the types that indicate a string is used to achieve the desired control state, this property holds the string to be used. This property is also used to hold the contents of a custom button assignment which calls a script when pressed.
ControlStringList (Array of String)	Similar to that of a value being capable of a range, a ControlStringList contains the list of string values which may be set upon the device's string to invoke a change. Use this when the list is dynamic and specific values cannot be assigned to status strings.
ControlFlag	Used only for rendering control options in a UI, this flag is set to True when a button is added to a device and indicates that a NewLine should be generated after the button so that the next button or control element can start on a new row.

See Also

clsValueRange

clsValueRange is an object used in the CAPIControl object to hold information about a RANGE value for a device. If, for example, a device can operate within a range from 1 to 99, you can denote this with a value/status pair that contains information about that range, rather than adding 99 value/status pair entries. A device that contains value/status pairs containing a range result in the CAPIControl object containing much the same information, including options for rendering the range properly on a user interface.

Here are the members of clsValueRange:

```
Public Class clsValueRange
    Inherits MarshalByRefObject
    Public RangeStart As Double
    Public RangeEnd As Double
    Public RangeStatusDecimals As Integer
    Public Property RangeStatusPrefix As String
    Public Property RangeStatusSuffix As String
    Public RangeStatusValueOffset As Double
    Public RangeStatusDivisor As Double
    Public ScaleReplace As String = ""
    Public HasScale As Boolean = False
End Class
```

The definition for each member is as follows:

Name	Description
RangeStart	This is the lowest value possible for this range definition.
RangeEnd	This is the highest value possible for this range definition.
RangeStatusDecimals	This value indicates how many decimal places the value range should be shown with. For example, if the range is 1 to 10 and the number of decimals is 1, then the actual range for purposes of display and selection is 1, 1.1, 1.2, 1.3... 9.8, 9.9, 10.
RangeStatusPrefix	This is a prefix to be placed in front of the value when displayed as a status. For example, if you set this to "Dim ", and the RangeStatusDecimals is 1, then the status when at the value 5.235689 will display as "Dim 5.2"
RangeStatusSuffix	This is a suffix to be appended to the end of the value when displayed as a status. For example, given the Prefix example above, set RangeStatusSuffix to "%" and the display will be "Dim 5.2%"
RangeStatusValueOffset	For situations where it is desired to have one range for control (set) and another for status (get), you can use this to indicate an offset from the value to get the desired display result. For example, if the range for controlling a device is 1 to 100 and the prefix/suffix is set to yield a control option such as "Set To 50 Percent" for the value 50, you can establish another range from 101 to 200 for status where the prefix is set to "Dimmed " and the suffix is set to "%", and the RangeStatusValueOffset is set to 100 such that when the value is set to 150, it results in "Dimmed 50%".
RangeStatusDivisor	This value is a divisor applied to the value before it is displayed. For example, if you have a hardware interface that produces values of 10,000 to 100,000, you may wish to represent this as "K" or Kilo rather than displaying all of the digits. To do this, set the suffix to "K" and set the RangeStatusDivisor to 1000. When the value is 55,555 it will result in a display of 55K, or if the RangeStatusDecimals are set to 2, the result would be 55.55K.
HasScale	When set to True, the range indicates that there is a ScaleReplace (scale replacement) indicator, and when the status is obtained, the scale text provided in the DeviceClass ScaleText property is inserted where the ScaleReplace is found.
ScaleReplace	When HasScale is True, HomeSeer will look for the string contained here and will replace it with the string value in the device's ScaleText property. For example, if you have a device that displays temperature but it is not known until runtime whether it will display in Celcius or Fahrenheit, set HasScale to True, and set ScaleReplace to a unique string such as "@S@". Set the suffix to " Degrees @S@", and then when the value is obtained from the device, set ScaleText to the proper scale such as "F", and when the status is obtained the result will be "xx Degrees F".

See Also

CAPIControlType
CAPIControlLocation

CAPIControlType

CAPIControlType is an Enum used within the CAPIControl object.

```
Enum CAPIControlType
    Not_Specified = 1
    Values = 2                    'This is the default to use if one of the others is not specified.
    Single_Text_from_List = 3
    List_Text_from_List = 4
    Button = 5
    ValuesRange = 6               'Rendered as a drop-list by default.
    ValuesRangeSlider = 7
    TextList = 8
    TextBox_Number = 9
    TextBox_String = 10
    Radio_Option = 11
End Enum
```

See Also

clsValueRange
CAPIControlLocation

CAPIControlLocation

This structure is part of the CAPIControl object, and holds the Row and Column values in a single structure.

```
Structure CAPIControlLocation
    Public Row As Integer
    Public Column As Integer
End Structure
```

See CAPIControl for more information regarding the use of these properties.

See Also

clsValueRange
CAPIControlType

CAPIControlHandler, CAPIControlsHandler

```
Public Function CAPIControlHandler(ByVal CC As CAPIControl) As CAPIControlResponse

Public Function CAPIControlslHandler(ByVal CC() As CAPIControl) As CAPIControlResponse
```

These functions are used to invoke a control method on a device using a CAPIControl object. Control of a single device to a single control state can be done using CAPIControlHandler, or several devices may be controlled at once using CAPIControlsHandler and passing an array of CAPIControl objects.

In both cases, the return is a single CAPIControlResponse Enum indicating the result of the operation.

See Also

CAPIGetStatus
CAPIGetControl, CAPIGetControlEx, CAPIGetSingleControl

CAPIControlResponse

CAPIControlResponse is an Enum which indicates the result of the CAPIControlHandler call made by a plug-in or script. It is defined as:

```
Public Enum CAPIControlResponse
    Indeterminate = 0
    All_Success = 1
    Some_Failed = 2
    All_Failed = 3
End Enum
```

See Also

Email

In This Section

MailDate
MailDelete
MailFrom
MailFromDisplay
MailMsgCount
MailSubject
MailText
MailTo
MailToDisplay
MailTrigger
SendEmail

See Also

MailDate

Purpose

This function returns the date of the indexed mail message.

Parameters

Parameter: **index**
Type: **integer**
Description: This is the index number of the message to be retrieved.

Returns

Return value: **date**
Type: **string**
Description: This is the date of the indexed mail message.

See Also

MailDelete
MailFrom
MailFromDisplay
MailMsgCount
MailSubject
MailText
MailTo
MailToDisplay
MailTrigger
SendEmail

MailDelete

Purpose

This function deletes the specified message.

Parameters

Parameter: **index**
Type: **integer**
Description: This is the index number of the message to be deleted.

Returns

None.

See Also

MailDate
MailFrom
MailFromDisplay
MailMsgCount
MailSubject
MailText
MailTo
MailToDisplay
MailTrigger
SendEmail

MailFrom

Purpose

This function returns the E-mail address of the person who sent the message.

Parameters

Parameter: **index**
Type: **integer**
Description: This is the index number of the message to be retrieved.

Returns

Return value: **from address**
Type: **string**
Description: This is the E-mail address of the sender for the indexed message.

See Also

MailDate
MailDelete
MailFromDisplay
MailMsgCount
MailSubject
MailText
MailTo
MailToDisplay
MailTrigger
SendEmail

MailFromDisplay

Purpose

This function returns the name of the person who sent the message.

Parameters

Parameter: **index**
Type: **integer**
Description: This is the index number of the message to be retrieved.

Returns

Return value: **from address**
Type: **string**
Description: This is the name of the sender for the indexed message.

See Also

MailDate
MailDelete
MailFrom
MailMsgCount
MailSubject
MailText
MailTo
MailToDisplay
MailTrigger
SendEmail

MailMsgCount

Purpose

This function returns the total number of unread messages that are in your Inbox. This will only work if MAPI is enabled as your E-mail interface (see the E-mail Setup screen).

Parameters

None.

Returns

Return value: **count**
Type: **integer**

See Also

MailDate
MailDelete
MailFrom
MailFromDisplay
MailSubject
MailText
MailTo
MailToDisplay
MailTrigger
SendEmail

MailSubject

Purpose

This function returns the subject of the specified message.

Parameters

Parameter: **index**
Type: **integer**

Description: This is the index number of the message to be retrieved.

Returns

Return value: **subject**
Type: **string**
Description: This is the subject of the indexed mail message.

See Also

MailDate
MailDelete
MailFrom
MailFromDisplay
MailMsgCount
MailText
MailTo
MailToDisplay
MailTrigger
SendEmail

MailText

Purpose

This function returns the body of the E-mail message.

Parameters

Parameter: **index**
Type: **integer**
Description: This is the index number of the message to be retrieved.

Returns

Return value: **body**
Type: **string**
Description: This is the body of the E-mail message.

See Also

MailDate
MailDelete
MailFrom
MailFromDisplay
MailMsgCount
MailSubject
MailTo
MailToDisplay
MailTrigger
SendEmail

MailTo

Purpose

This function returns the E-mail address of the person the message was sent to.

Parameters

Parameter: **index**
Type: **integer**
Description: This is the index number of the message to be retrieved.

Returns

Return value: **to address**
Type: **string**
Description: This is the E-mail address of the recipient of the indexed message.

See Also

MailDate
MailDelete
MailFrom
MailFromDisplay
MailMsgCount
MailSubject
MailText
MailToDisplay
MailTrigger
SendEmail

MailToDisplay

Purpose

This function returns the name of the person the message was sent to.

Parameters

Parameter: **index**
Type: **integer**
Description: This is the index number of the message to be retrieved.

Returns

Return value: **name**
Type: **string**
Description: This is the name of the recipient.

See Also

MailDate
MailDelete
MailFrom
MailFromDisplay
MailMsgCount
MailSubject
MailText
MailTo
MailTrigger
SendEmail

MailTrigger

Purpose

This function returns the index in the MAPI message list of the message that caused the last trigger. If you created an event that triggers when an E-mail is received, then you can run a script as the action to the event. In the script, you call this function to get the index for the message that caused the trigger. You can now examine the message to take more action.

Parameters

None.

Returns

Return value: **index**
Type: **long**
Description: This is the index of the E-mail message that was last received.

Example

```
              ' this script will access the E-mail message that caused this event to trigger
' it will then speak the subject line of the message

sub main()

     dim index
     dim subject
     index = hs.MailTrigger
     subject = hs.MailSubject(index)
     hs.speak "You have mail! The subject is "
     hs.speak subject

end sub
```

See Also

MailDate
MailDelete
MailFrom
MailFromDisplay
MailMsgCount
MailSubject
MailText
MailTo
MailToDisplay
SendEmail

SendEmail

Purpose

This function will send an E-mail message.

The `attach` parameter is useful if you would like to E-mail a picture taken with a digital camera. Just attach the path to any picture file created by the camera.

Parameters

Parameter: **mto**
Type: **string**
Description: This is the address you are sending the E-mail to.

Parameter: **mfrom**
Type: **string**
Description: This is the address you are sending from. Note that some ISPs will not allow you to put just anything in this field. You may be required to put your real E-mail address here. If you are using MAPI to handle your E-mail, MAPI will enter your E-mail address that is associated with your default E-mail account. In that case, this field will be ignored.

parameter:**mCC**
Type: **string**
Description: CC address

Parameter: **mBCC**
Type: **string**
Description: This is BCC to address.

Parameter: **msubject**
Type: **string**
Description: This is the subject of the E-mail.

Parameter: **message**
Type: **string**
Description: This is the body of the E-mail.

Parameter: **attach** (optional)
Type: **string**
Description: This is the absolute path name to the file to be attached to the E-mail.

Returns

None.

See Also

MailDate
MailDelete
MailFrom
MailFromDisplay
MailMsgCount
MailSubject
MailText
MailTo
MailToDisplay
MailTrigger

Events

In This Section

Get Information
Get Event References
Modify Automatic Triggering
Triggering Events
Modifying Events
SetSecurityMode

See Also

About Scripts
Applications and Plugins
Computer
Devices
Email
Internet
Phone
Scripts
Speech Recognition
Strings, Global Variables, and Encryption
Time and Calendar
Text-To-Speech and Media

Get Information

In This Section

See Also

Event_Group_Info_All

```
Function Event_Group_Info_All() As strEventGroupData()
```

Purpose

This function returns information about all of the event groups in the system.

Parameters

Parameter: **None**

Returns

Return value: **strEventGroupData**
Type: **array of structure**
Description: This structure is described in this section.

See Also

strEventGroupData

This structure is used as a single return value or an array return value for functions that request information about event groups. The description of each member is below.

```
Public Structure strEventGroupData
    Public GroupID As Integer     ' This is the event group reference ID.
    Public GroupName As String    ' The name of the group.
    Public Global_Actions_Count As Integer ' The number of global actions in this event group.
    Public Global_Actions As String()    ' The list (array) of global actions in action_type : action_name format.
    Public Global_Conditions_Count As Integer ' The number of global conditions in this event group.
    Public Global_Conditions As String()  ' The list (array) of global conditions in trigger_type : trigger_name format.
End Structure
```

See Also

Event_Group_Info

```
Function Event_Group_Info(ByVal GroupRef As Integer) As strEventGroupData
```

Purpose

This function returns information about a single event group using its Group Reference ID number.

Parameters

Parameter: **GroupRef**
Type: **integer**
Description: This is the group reference ID number for the event group you want to return information about.

Returns

Return value: **strEventGroupData**
Type: **structure**
Description: This structure is described in this section.

See Also

Event_Group_Info_All
Event_Info_All
Event_Info
Event_Info_Group
EventCount
EventExists
GetLastEvent

strEventGroupData

This structure is used as a single return value or an array return value for functions that request information about event groups. The description of each member is below.

```
Public Structure strEventGroupData
    Public GroupID As Integer     ' This is the event group reference ID.
    Public GroupName As String    ' The name of the group.
    Public Global_Actions_Count As Integer ' The number of global actions in this event group.
    Public Global_Actions As String()    ' The list (array) of global actions in action_type : action_name format.
    Public Global_Conditions_Count As Integer ' The number of global conditions in this event group.
    Public Global_Conditions As String()  ' The list (array) of global conditions in trigger_type : trigger_name format.
End Structure
```

See Also

Event_Info_All

```
Function Event_Info_All() As strEventData()
```

Purpose

This function returns information about all events in the system.

Parameters

Parameter: **None**

Returns

Return value: **strEventData**
Type: **array of structure**
Description: This structure is described in this section.

See Also

Event_Group_Info_All
Event_Group_Info
Event_Info
Event_Info_Group
EventCount
EventExists
GetLastEvent

strEventData

This structure is used as a single return or array return from functions providing information about events. Descriptions of each member of the structure are below.

```
Public Structure strEventData
    Public Event_Ref As Integer     ' The event reference ID number.
    Public Event_Name As String     ' The event name
    Public Event_Type As String     ' The event type, if used.
```

```
    Public GroupID As Integer      ' The event group reference ID number.
    Public GroupName As String     ' The event group name.
    Public UserNote As String      ' The user's note contents.
    Public Last_Triggered As Date     ' The time the event was last triggered or
    '          Date.MinValue if it has not been triggerd before.
    Public Retrigger_Delay As TimeSpan    ' If the event is prevented from triggering within a given amount of time,
    '          this timespan will contain that time period.
    Public Flag_Enabled As Boolean     ' True if the event is enabled for automatic triggering.
    Public Flag_Delete_After_Trigger As Boolean  ' True if the event is deleted from the system after it triggers.
    Public Flag_Do_Not_Log As Boolean    ' True if the event is set to not log information when it is triggered.
    Public Flag_Delayed_Event As Boolean    ' True if the event was created as a result of a delayed action or
trigger.
    Public Flag_Include_in_Powerfail As Boolean  ' True if the event is to be included in powerfailure recovery.
    Public Flag_Security As Boolean     ' True if the event trigger(s) can be modified by a random amount
    '          when the security feature is enabled.
    Public Flag_Priority_Event As Boolean    ' True if the event is set to not have its execution queued.
    Public Action_Count       ' The number of actions in this event.
    Public Actions As String()      ' The list (array) of actions in action_type : action_name format.
    Public Trigger_Count As Integer     ' The total number of triggers and conditions in this event.
    Public Trigger_Group_Count As Integer    ' The number of trigger groups (If / Or If) in the event.
    Public Trigger_Groups As strEventTriggerGroupData() ' The list (array of structure) of triggers in each trigger
group.
End Structure
```

See Also

strEventTriggerGroupData

strEventTriggerGroupData

This structure, which is used within the strEventData structure, is used by functions that return information about events. The description of each member is below.

```
Public Structure strEventTriggerGroupData
    Public GroupNumber As Integer    ' The trigger group number for this group of triggers and conditions.
    Public Triggers As String()     ' The list (array) of triggers in this group in trigger_type : trigger_name format.
End Structure
```

See Also

strEventData

Event_Info

```
Function Event_Info(ByVal evRef As Integer) As strEventData
```

Purpose

This function returns information about a single event using its Event Reference ID number.

Parameters

Parameter: **evRef**
Type: **integer**
Description: This is the event reference ID number for the event you want to return information about.

Returns

Return value: **strEventData**
Type: **structure**
Description: This structure is described in this section.

See Also

Event_Group_Info_All
Event_Group_Info
Event_Info_All
Event_Info_Group
EventCount
EventExists
GetLastEvent

strEventData

This structure is used as a single return or array return from functions providing information about events. Descriptions of each member of the structure are below.

```
Public Structure strEventData
    Public Event_Ref As Integer     ' The event reference ID number.
    Public Event_Name As String     ' The event name
    Public Event_Type As String     ' The event type, if used.
    Public GroupID As Integer       ' The event group reference ID number.
    Public GroupName As String      ' The event group name.
    Public UserNote As String       ' The user's note contents.
    Public Last_Triggered As Date     ' The time the event was last triggered or
    '           Date.MinValue if it has not been triggerd before.
    Public Retrigger_Delay As TimeSpan   ' If the event is prevented from triggering within a given amount of time,
    '           this timespan will contain that time period.
    Public Flag_Enabled As Boolean      ' True if the event is enabled for automatic triggering.
    Public Flag_Delete_After_Trigger As Boolean  ' True if the event is deleted from the system after it triggers.
    Public Flag_Do_Not_Log As Boolean    ' True if the event is set to not log information when it is triggered.
    Public Flag_Delayed_Event As Boolean   ' True if the event was created as a result of a delayed action or
trigger.
    Public Flag_Include_in_Powerfail As Boolean  ' True if the event is to be included in powerfailure recovery.
    Public Flag_Security As Boolean     ' True if the event trigger(s) can be modified by a random amount
    '           when the security feature is enabled.
    Public Flag_Priority_Event As Boolean   ' True if the event is set to not have its execution queued.
    Public Action_Count       ' The number of actions in this event.
    Public Actions As String()      ' The list (array) of actions in action_type : action_name format.
    Public Trigger_Count As Integer     ' The total number of triggers and conditions in this event.
```

```
    Public Trigger_Group_Count As Integer   ' The number of trigger groups (If / Or If) in the event.
    Public Trigger_Groups As strEventTriggerGroupData() ' The list (array of structure) of triggers in each trigger group.
End Structure
```

See Also

strEventTriggerGroupData

strEventTriggerGroupData

This structure, which is used within the strEventData structure, is used by functions that return information about events. The description of each member is below.

```
Public Structure strEventTriggerGroupData
    Public GroupNumber As Integer   ' The trigger group number for this group of triggers and conditions.
    Public Triggers As String()     ' The list (array) of triggers in this group in trigger_type : trigger_name format.
End Structure
```

See Also

strEventData

Event_Info_Group

```
Function Event_Info_Group(ByVal GroupID As Integer) As strEventData()
```

Purpose

This function returns information about all of the events within an event group using the Event Group Reference ID number.

Parameters

Parameter: **GroupID**
Type: **integer**
Description: This is the event group reference ID number for the event group you want to return information about.

Returns

Return value: **strEventData**
Type: **structure**
Description: This structure is described in this section.

Note:

Use Event_Group_Info_All to find the GroupID for the event group you want to get event information about.

See Also

Event_Group_Info_All
Event_Group_Info
Event_Info_All
Event_Info
EventCount
EventExists
GetLastEvent

strEventData

This structure is used as a single return or array return from functions providing information about events. Descriptions of each member of the structure are below.

```
Public Structure strEventData
    Public Event_Ref As Integer     ' The event reference ID number.
    Public Event_Name As String     ' The event name
    Public Event_Type As String     ' The event type, if used.
    Public GroupID As Integer       ' The event group reference ID number.
    Public GroupName As String      ' The event group name.
    Public UserNote As String       ' The user's note contents.
    Public Last_Triggered As Date     ' The time the event was last triggered or
    '           Date.MinValue if it has not been triggerd before.
    Public Retrigger_Delay As TimeSpan    ' If the event is prevented from triggering within a given amount of time,
    '           this timespan will contain that time period.
    Public Flag_Enabled As Boolean     ' True if the event is enabled for automatic triggering.
    Public Flag_Delete_After_Trigger As Boolean  ' True if the event is deleted from the system after it triggers.
    Public Flag_Do_Not_Log As Boolean    ' True if the event is set to not log information when it is triggered.
    Public Flag_Delayed_Event As Boolean    ' True if the event was created as a result of a delayed action or
trigger.
    Public Flag_Include_in_Powerfail As Boolean  ' True if the event is to be included in powerfailure recovery.
    Public Flag_Security As Boolean     ' True if the event trigger(s) can be modified by a random amount
    '           when the security feature is enabled.
    Public Flag_Priority_Event As Boolean    ' True if the event is set to not have its execution queued.
    Public Action_Count       ' The number of actions in this event.
    Public Actions As String()      ' The list (array) of actions in action_type : action_name format.
    Public Trigger_Count As Integer     ' The total number of triggers and conditions in this event.
    Public Trigger_Group_Count As Integer    ' The number of trigger groups (If / Or If) in the event.
    Public Trigger_Groups As strEventTriggerGroupData() ' The list (array of structure) of triggers in each trigger
group.
End Structure
```

See Also

strEventTriggerGroupData

strEventTriggerGroupData

This structure, which is used within the strEventData structure, is used by functions that return information about events. The description of each member is below.

```
Public Structure strEventTriggerGroupData
    Public GroupNumber As Integer    ' The trigger group number for this group of triggers and conditions.
    Public Triggers As String()    ' The list (array) of triggers in this group in trigger_type : trigger_name format.
End Structure
```

See Also

strEventData

EventCount

Purpose

This function returns the total number of events configured in the system.

Parameters

None.

Returns

Return value: **value**
Type: **integer**

See Also

Event_Group_Info_All
Event_Group_Info
Event_Info_All
Event_Info
Event_Info_Group
EventExists
GetLastEvent

EventExists

Purpose

This function checks if a given events exists in HomeSeer's event list.

Parameters

Parameter: **name**

Type: **string**
Description: This is the name of the event. The name is not case-sensitive.

Returns

Return value: **event index**
Type: **boolean**
Description: This returns TRUE if the given event exists and FALSE if it doesn't.

Example

```
sub main()

      if hs.EventExists("evening") then

            hs.speak "The evening event exists",TRUE

      else

            hs.speak "The evening event does not exist",TRUE

      end if

end sub
```

See Also

Event_Group_Info_All
Event_Group_Info
Event_Info_All
Event_Info
Event_Info_Group
EventCount
GetLastEvent

GetLastEvent

Purpose

This function returns the name of the last event that was triggered. This can be used in a script to detect which event the script was executed from.

Parameters

None.

Returns

Return value: **last event**
Type: **string**

Example

```
sub main()

      dim t

      t = hs.GetLastEvent

      msgbox "This script is run from the event: " & t

end sub
```

See Also

Event_Group_Info_All
Event_Group_Info
Event_Info_All
Event_Info
Event_Info_Group
EventCount
EventExists

Home > Scripting > Events > Get Event References

Get Event References

In This Section

GetEventEx
GetEventByRef
GetEventRefByName

See Also

Get Information
Modify Automatic Triggering
Triggering Events
Modifying Events
SetSecurityMode

Home > Scripting > Events > Get Event References > GetEventEx

GetEventEx

GetEventEx

Purpose

This function returns a reference to an event with the given name. The returned object can be used to access properties of the event.

- If multiple events have the same name, the wrong event may be returned.

Parameters

Parameter: **event name**
Type: **string**
Description: This is the name of the event.

Returns

Return value: **event reference**
Type: **object as EventClass**

Example

```
dim ev

set ev = hs.GetEventEx("night event")

msgbox "The event type is: " & cstr(ev.ev_abs_time)
```

See Also

GetEventByRef
GetEventRefByName

GetEventByRef

GetEventByRef

Purpose

This function returns an event. The event is of type `EventClass` and can be used to retrieve and set properties of an event.

An `EventClass` has a number of properties that holds information about an event. You can access these properties to get and set this information. The function NewEventGetRef can be used to create a new empty event. The properties of the event are listed below.

Parameters

Parameter: **event reference ID**
Type: **long (.NET Integer)**
Description: This is the event reference ID of the desired event.

Returns

Return value: **EventClass**
Type: **object**

EventClass Object

See EventClass

See Also

GetEventEx
GetEventRefByName

GetEventRefByName

GetEventRefByName

Purpose

This function returns the event reference for an event. The event reference is different than an index to a event. The event reference is only needed for other procedures which explicitly require the event reference ID.

- This will only return the reference to the first event matching the name provided.

Parameters

Parameter: **sName**
Type: **string**
Description: This is the event name excluding the group, such as "Wake-Up Time".

Returns

Return value: **reference**
Type: **long (.NET Integer)**
Description: This is a numerical event reference ID.

See Also

GetEventEx
GetEventByRef

Modify Automatic Triggering

In This Section

EnableEvent
EnableEventByRef
DisableEvent
DisableEventByRef

See Also

Get Information
Get Event References
Triggering Events
Modifying Events
SetSecurityMode

EnableEvent

Purpose

This function marks an event as enabled. All triggers are active.

Parameters

Parameter: **evname**
Type: **string**
Description: This is the event name to enable. Note that the name is not case-sensitive, and the event must have already been disabled.

Returns

None.

See Also

EnableEventByRef
DisableEvent
DisableEventByRef

EnableEventByRef

Purpose

This function marks an event as enabled. All triggers are active.

Parameters

Parameter: **evref**
Type: **long (.NET Integer)**
Description: This is the event reference ID of the event to enable.

Returns

None.

Example

```
sub main()

      dim eref

      eref = hs.GetEventRefByName("My Event")
      hs.EnableEventByRef eref

end sub
```

See Also

EnableEvent
DisableEvent
DisableEventByRef

DisableEvent

Purpose

This function marks an event as disabled. All triggers are suspended until the event is re-enabled.

Parameters

Parameter: **evname**
Type: **string**
Description: This is the event name to disable. Note that the name is not case-sensitive

Returns

None.

Example

```
sub main()

      hs.DisableEvent "evening"

end sub
```

See Also

EnableEvent
EnableEventByRef
DisableEventByRef

DisableEventByRef

Purpose

This function marks an event as disabled. All triggers are suspended until the event is re-enabled.

Parameters

Parameter: **evref**
Type: **long (.NET Integer)**
Description: This is the event reference ID of the event to be disabled.

Returns

None.

Example

```
sub main()

      dim eref

      eref = hs.GetEventRefByName("My Event")
      hs.DisableEventByRef eref

end sub
```

See Also

EnableEvent
EnableEventByRef
DisableEvent

Triggering Events

In This Section

TriggerEvent
TriggerEventEx
DelayTrigger
TriggerEventAndWait
RemoveDelayedEvent

See Also

Get Information
Get Event References
Modify Automatic Triggering
Modifying Events
SetSecurityMode

TriggerEvent

Purpose

This function forces an event to be triggered.

Parameters

Parameter: **name**
Type: **string**
Description: This is the name of the event you want to trigger. The actions for the event are executed. Note that the name is not case-sensitive. For instance, the events "Evening" and "evening" would be considered the same.

- If there were duplicate event names, only the first one found would run.

Returns

Return value: **status**
Type: **integer**
Description: This is 0 if there was an error or 1 if there was no error. If an error is detected, then an error message is written to the event log.

Example

```
'Trigger the event named "turn all lights on"

sub main()
                hs.TriggerEvent "turn all lights on"
end sub
```

See Also

TriggerEventEx
DelayTrigger
TriggerEventAndWait
RemoveDelayedEvent

TriggerEventEx

Purpose

This function forces an event to be triggered and is used instead of TriggerEvent when it is necessary to specify phone line information at the same time.

Parameters

Parameter: **line**
Type: **integer**
Description: This is the phone line number that you wish the event to have been triggered from.

Parameter: **name**
Type: **string**
Description: This is the name of the event you want to trigger. The actions for the event are executed. Note that the name is not case-sensitive. For instance, the events "Evening" and "evening" would be considered the same.

- If there were duplicate event names, only the first one found would run.

Parameter: **voice command (Optional)**
Type: **string**
Description: If the event processes voice commands, you can provide the string of what the recognized voice command would be that you want the event to process.

Returns

Return value: **status**
Type: **integer**
Description: This is 0 if there was an error or 1 if there was no error. If an error is detected, then an error message is written to the event log.

Example

```
'Trigger the event named "turn all lights on"
```

```
sub main()
  hs.TriggerEvent 1, "turn all lights on"
end sub
```

See Also

TriggerEvent
DelayTrigger
TriggerEventAndWait
RemoveDelayedEvent

DelayTrigger

Purpose

This function triggers the given event after the specified number of seconds have elapsed. This is handy if you would like to turn a device on or off a few seconds after the initial event triggers. Note that you can call this as many times as you like, as new events are created and may be viewed and deleted from your events view.

Parameters

Parameter: **secs**
Type: **long**
Description: This is the number of seconds before the event name `evname` is triggered.

Parameter: **evname**
Type: **string**
Description: This is the text name of the event that will be triggered.

Returns

None.

Example

```
sub main()

     ' delay the execution of the event named "lights off" by 5 minutes

     hs.DelayTrigger 300, "lights off"

end sub
```

See Also

TriggerEvent
TriggerEventEx
TriggerEventAndWait
RemoveDelayedEvent

TriggerEventAndWait

Purpose

This function forces an event to be triggered and does not return until the event has completed.

Parameters

Parameter: **name**
Type: **string**
Description: This is the name of the event you want to trigger. The actions for the event are executed. Note that the name is not case-sensitive. For instance, the events "Evening" and "evening" would be considered the same.

- If there were duplicate event names, only the first one found would run.

Returns

Return value: **status**
Type: **integer**
Description: This is 0 if there was an error or 1 if there was no error. If an error is detected, then an error message is written to the event log.

Example

Trigger the event named "turn all lights on" using a VB.NET script.

```
iReturn = hs.TriggerEventAndWait("turn all lights on")
If iReturn = 0 Then
   hs.WriteLog("Error","There was an error triggering the event: Turn All Lights On")
End If
```

See Also

TriggerEvent
TriggerEventEx
DelayTrigger
RemoveDelayedEvent

RemoveDelayedEvent

Purpose

This function removes a previously queued event from the pending event queue. There are two types of pending events. The first is a device operation and the pending event contains only the house code and unit code of the device. The second is the queuing of an actual event name. In this case, only the name of the event is in the queue. This is typically queued from the DelayTrigger script function.

When a delayed action is used on a device, it will appear under the group "Delayed Actions" in HomeSeer's Events screen.

Parameters

Parameter: **device**
Type: **string**
Description: This is the X10 code of the device like "A1".

Parameter: **event_name**
Type: **string**
Description: This is the name of the event that is to be removed from the queue.

Returns

None.

Example

Here is a sample of how to use RemoveDelayedEvent to remove a queued device action for the device at address A7:

```
hs.RemoveDelayedEvent "A7",""
```

Here is a sample of how to use RemoveDelayedEvent to remove a queued event action for the event named "Reset Dryer Reminder":

```
hs.RemoveDelayedEvent "","Reset Dryer Reminder"
```

See Also

TriggerEvent
TriggerEventEx
DelayTrigger
TriggerEventAndWait

Modifying Events

In This Section

See Also

Get Information
Get Event References
Modify Automatic Triggering
Triggering Events
SetSecurityMode

AddDeviceActionToEvent

```
Public Function AddDeviceActionToEvent(ByVal evRef As Integer, ByVal CC As CAPIControl) As String
```

Purpose

This procedure will add a device action to an existing event.

Parameters

Parameter: **evRef**
Type: **Integer**
Description: This is the event reference ID number of the event you wish to add a device action to.

Parameter: **CC**
Type: **CAPIControl**
Description: The CAPIControl options for a device can be obtained by using CAPIGetControl . Once the CAPIControls of a device are retrieved, find the desired control action, and use that as the CC parameter to this function to have that device action added to the event.

Returns

Return value: **Result**
Type: **String**
Description: When empty, the procedure was successful. If this string is not empty, it will contain information about the failure encountered with this function call.

Example

The following script will ...

See Also

EventSetTimeTrigger
EventSetRecurringTrigger
NewEventEx
NewEventGetRef
SaveEventsDevices
DeleteEvent

EventSetTimeTrigger

```
Public Function EventSetTimeTrigger(ByVal evRef As Integer, ByVal DT As Date) As Boolean
```

Purpose

This procedure will set a time trigger on an event. The trigger is always the first trigger in the first trigger group. If the event already has a trigger in the first trigger group, the group will be wiped out and replaced with this time trigger. The date component of the date parameter is ignored.

Parameters

Parameter: **evRef**
Type: **Integer**
Description: This is the event reference ID number for the event you wish to set a time trigger on.

Parameter: **DT**
Type: **Date**
Description: This DATE data type is used to set the time you wish to trigger the event at. The date part of the DT parameter is ignored.

Returns

Return value: **Result**
Type: **Boolean**
Description: When True, the procedure was successful.

Example

See Also

AddDeviceActionToEvent
EventSetRecurringTrigger
NewEventEx
NewEventGetRef
SaveEventsDevices
DeleteEvent

EventSetRecurringTrigger

```
Public Function EventSetRecurringTrigger(ByVal evRef As Integer, ByVal Frequency As TimeSpan, _
                                         ByVal Once_Per_Hour As Boolean, _
                                         ByVal Reference_To_Hour As Boolean) As Boolean
```

Purpose

This procedure sets the trigger on an existing event to be a recurring time trigger.

Parameters

Parameter: **evRef**
Type: **Integer**
Description: This is the event reference ID number of the event that you wish to set to a recurring trigger.

Parameter: **Frequency**
Type: **TimeSpan**
Description: The time period held within this parameter will be the recurrence frequency for the event. The TimeSpan should not have any value for the Days part and the Hours part value should be less than 24.

Parameter: **Once_Per_Hour**
Type: **Boolean**
Description: If set to True, the event will only trigger once per hour even if the Frequency is less than one hour.

Parameter: **Reference_To_Hour**
Type: **Boolean**
Description: When set to True, the Frequency will be calculated from the top of the hour rather than the previous trigger time or the first trigger time.

Returns

Return value: **Result**
Type: **Boolean**
Description: When True, the procedure was successful.

Example

See Also

AddDeviceActionToEvent
EventSetTimeTrigger
NewEventEx
NewEventGetRef
SaveEventsDevices
DeleteEvent

NewEventEx

```
Public Function NewEventEx(ByVal Name As String, ByVal Group As String, ByVal sType As String) As Integer
```

```
Public Function NewEventGetRef(ByVal Name As String, ByVal Group As String, ByVal sType As String) As Integer
```

Purpose

This function creates a new empty event. The trigger mode is set to MANUAL and the event is DISABLED if the setup option "New Events are Disabled by Default" is enabled. All other properties are cleared and the name is set to the name given. The EventClass object reference to the new event is returned and may be used in a script to set properties of the new event.

Parameters

Parameter: **Name**
Type: **string**
Description: This is the name of the new event.

Parameter: **Group**
Type: **string**
Description: This is the name of the event group you want the new event created in. If the event group does not exist, it will be created.

Parameter: **sType**
Type: **string**
Description: This is the type description for the event that you wish to use.

Returns

Return value: **event reference**
Type: **Integer**
Description: This is the event reference ID number.

Example

See Also

AddDeviceActionToEvent
EventSetTimeTrigger
EventSetRecurringTrigger
NewEventGetRef
SaveEventsDevices
DeleteEvent

NewEventGetRef

```
Public Function NewEventEx(ByVal Name As String, ByVal Group As String, ByVal sType As String) As Integer
```

```
Public Function NewEventGetRef(ByVal Name As String, ByVal Group As String, ByVal sType As String) As Integer
```

Purpose

This function creates a new empty event. The trigger mode is set to MANUAL and the event is DISABLED if the setup option "New Events are Disabled by Default" is enabled. All other properties are cleared and the name is set to the name given. The EventClass object reference to the new event is returned and may be used in a script to set properties of the new event.

Parameters

Parameter: **Name**
Type: **string**
Description: This is the name of the new event.

Parameter: **Group**
Type: **string**
Description: This is the name of the event group you want the new event created in. If the event group does not exist, it will be created.

Parameter: **sType**
Type: **string**
Description: This is the type description for the event that you wish to use.

Returns

Return value: **event reference**
Type: **Integer**
Description: This is the event reference ID number.

Example

See Also

AddDeviceActionToEvent
EventSetTimeTrigger
EventSetRecurringTrigger
NewEventEx
SaveEventsDevices
DeleteEvent

SaveEventsDevices

SaveEventsDevices

Purpose

If an event or device was modified by a script, this function should be called to update HomeSeer with the changes. For example, if you change a voice command in an event, calling this function tells HomeSeer to re-initialize the voice recognition so the new voice command is available to the user. This function will also update all displays in the Control screen.

Parameters

None.

Returns

None.

See Also

AddDeviceActionToEvent
EventSetTimeTrigger
EventSetRecurringTrigger
NewEventEx
NewEventGetRef
DeleteEvent

DeleteEvent

DeleteEvent

Purpose

This function deletes the specified event.

Parameters

Parameter: **evname**
Type: **string**
Description: This is the event name to delete. Note that the name is not case-sensitive.

Returns

None.

Example

```
sub main()

     hs.DeleteEvent "evening"

end sub
```

See Also

AddDeviceActionToEvent
EventSetTimeTrigger
EventSetRecurringTrigger
NewEventEx
NewEventGetRef
SaveEventsDevices

SetSecurityMode

Purpose

This function enables or disables the security mode. When security mode is enabled, the event trigger time is randomly set to plus or minus 30 minutes from the actual set time. This can give your home a lived-in look because lights and other devices won't be turned on at the same time day after day.

Parameters

Parameter: **mode**
Type: **integer**
Description: Use 0 to disable the security mode and 1 to enable it.

Returns

None.

Example

The following script statement will enable security mode.

```
hs.SetSecurityMode 1
```

See Also

Get Information
Get Event References
Modify Automatic Triggering
Triggering Events
Modifying Events

Internet

In This Section

FTP
GetURL
GenCookieString

See Also

FTP

In This Section

See Also

GetURL
GenCookieString

FTP

Purpose

This function gives access to ftp servers. This command is used mostly for downloading files using the ftp protocol. Use FTPLastError to check for errors executing this command.

When using the `get` and `put` commands, the `local_file` and `remote_file` parameters must be valid. For a `put` command, the command will copy the file at the path given in the `local_file` parameter to the path given in the `remote_file` parameter. Note that the `remote_file` specification should not include any path information. Set the `path` parameter to the correct path for the file. For `get` commands, the file at the `remote_file` location is copied to the file at the `local_file` location.

For the `del` command, the file at the `remote_file` location is deleted.

The `dir` command returns the directory as a string.

The rename command uses `local_file` as the old name, and `remote_file` as the new name.

Parameters

Parameter: **host**
Type: **string**
Description: This is the name or IP address of host to connect to, such as `HomeSeer.com`.

Parameter: **username**
Type: **string**
Description: This is the username for access to the server.

Parameter: **password**
Type: **string**
Description: This is the password for access to the server.

Parameter: **command**
Type: **string**

Description: This can be one of the following FTP commands: `put`, `get`, `del`, `dir`, or `rename`.

Parameter: **path**
Type: **string**
Description: This is the path to the file, such as `public`.

Parameter: **local_file**
Type: **string**
Description: This is the file name where the downloaded file will be saved.

Parameter: **remote_file**
Type: **string**
Description: This is the name of the file on the remote server to download.

Returns

Return value: **depends on command**
Type: **string**
Description: For the `dir` command, a directory listing is returned. For all other commands, if no error occurs an empty string is returned, else an error message is returned which starts with the text "ERROR".

Example

```
sub main()

     dim s
     dim host
     dim user
     dim password
     dim command
     dim rfile
     dim lfile
     dim path

     host = "homeseer.com"
     user = "anonymous"
     password = "user@company.com"
     command = "get"
     rfile = "remote_test.htm"
     lfile = "c:\remote_test.htm"
     path = "pub"

     ' get the file
     s = hs.ftp(host,user,password,command,path,lfile,rfile)

end sub
```

See Also

FTPLastError
SetRemoteTimeout

FTPLastError

Purpose

This command should be used after an FTP command to check for any errors that may have been encountered with the FTP command. A null (empty) return indicates that the command completed successfully.

Parameters

None.

Returns

Return value: **Error**
Type: **string**
Description: This is the text of the last FTP command error.

See Also

FTP
SetRemoteTimeout

SetRemoteTimeout

Purpose

This function sets the number of seconds to wait for a remote host to respond when using hs.GetURL, hs.GetURLIE, or hs.ftp.

- If this function is never called, the remote timeout is set to 60 seconds.

Parameters

Parameter: **timeout**
Type: **integer**
Description: This is the number of seconds to wait.

Returns

None.

Example

```
' set the remote timeout to 30 seconds

hs.SetRemoteTimeout 30
```

See Also

FTP
FTPLastError

GetURL

In This Section

GetURL
GetURLEx
GetURLIE
GetURLImage
GetURLImageEx
URLAction
SetRemoteTimeout

See Also

FTP
GenCookieString

GetURL

Purpose

This function returns a web page. This is useful for retrieving pages like news and weather, and then having a Speaker Client speak the contents for you. This method can also be used to retrieve images, including JPG or GIF images.

Parameters

Parameter: **host**
Type: **string**
Description: This is the name or IP address of host to connect to, such as "HomeSeer.com".

Parameter: **page**
Type: **string**
Description: This is the page to retrieve from the server, such as "/news.htm".

Parameter: **strip_tags**
Type: **boolean**
Description: Use TRUE to strip HTML tags from the returned page or FALSE to not alter the page.

Parameter: **port**
Type: **integer**
Description: This is the port number on the server to connect with (80 = standard web server).

Optional Parameter: **UTF8**
Type: **boolean**
Description: When set to TRUE, the HomeSeer will decode the data received from the web server using UTF-8. If this is FALSE or is not provided, then the data will be decoded using the default encoding (usually Windows-1252).

Returns

Return value: **page contents**
Type: **string**
Description: This is the contents of the requested web page. If an error occurs, the text "ERROR:" will be returned followed by a reason for the error.

Example

```
sub main()

     dim page

     page = hs.GetURL("HomeSeer.com","/",TRUE,80)
     msgbox page

               end sub
```

See Also

GetURLEx
GetURLIE
GetURLImage
GetURLImageEx
URLAction
SetRemoteTimeout

GetURLEx

Purpose

This function returns a web page. This is useful for retrieving pages like news and weather, and then having a Speaker Client speak the contents for you. This method can also be used to retrieve images, including JPG or GIF images. This function has extended features compared to the similar function GetURL - it can return a byte array, which is useful when retrieving binary data, and it attempts to decode the page data encoding so that the data is properly decoded using Windows-1252 or UTF-8.

Parameters

Parameter: **host**
Type: **string**
Description: This is the name or IP address of host to connect to, such as "HomeSeer.com".

Parameter: **page**
Type: **string**
Description: This is the page to retrieve from the server, such as "/news.htm".

Parameter (ByRef): **ElapsedTime**
Type: **string**
Description: Provide an empty string variable for this parameter, and when the procedure is finished, it will contain a formatted string with the total time the page download required.

Optional Parameter: **port**
Type: **integer**
Description: This is the port number on the server to connect with (80 = standard web server). If this parameter is not provided, a value of 80 will be used.

Optional Parameter: **strip_tags**
Type: **boolean**
Description: Use TRUE to strip HTML tags from the returned page or FALSE to not alter the page.

Optional Parameter: **ByteArray**
Type: **boolean**
Description: If set to TRUE, then the return from the function will be an array of bytes instead of a string.

Optional Parameter: **FileName**
Type: **string**
Description: If this parameter is not null or empty, then the downloaded web page will be automatically saved into the file named with this parameter. HomeSeer must have write access to the directory where the file is to be placed. An existing file by the same name will be OVERWRITTEN.

Returns

Return value: **page contents**
Type: **string** or **byte array**
Description: This is the contents of the requested web page. If an error occurs, the text "ERROR:" will be returned followed by a reason for the error.

See Also

GetURL
GetURLIE
GetURLImage
GetURLImageEx
URLAction
SetRemoteTimeout

GetURLIE

Purpose

This function returns a web page using Internet Explorer. Note that only the HTML of the page is returned, but the entire page will be downloaded from the specified website. If the page contains any sounds, the sounds may be played out your computer speakers. Try using hs.GetURL before using this function.

Parameters

Parameter: **host**
Type: **string**
Description: This is the name or IP address of host to connect to, such as "HomeSeer.com".

Parameter: **strip_tags**
Type: **boolean**
Description: Use TRUE to strip HTML tags from the returned page or FALSE to not alter the page.

Returns

Return value: **page contents**

Type: **string**
Description: This is the contents of the requested web page. If an error occurs, the text "ERROR:" will be returned followed by a reason for the error.

See Also

GetURL
GetURLEx
GetURLImage
GetURLImageEx
URLAction
SetRemoteTimeout

GetURLImage

Purpose

This function returns a web page image file. This is useful for retrieving pages like weather satellite maps, and then displaying the maps in a HomeSeer device.

Parameters

Parameter: **host**
Type: **string**
Description: This is the name or IP address of host to connect to, such as "HomeSeer.com".

Parameter: **page**
Type: **string**
Description: This is the image to retrieve from the server, such as "/logo.gif". It should be fully qualified as referenced from the host parameter above, such as "\images\something\other\logo.gif" if necessary.

Optional Parameter: **strip_tags**
Type: **boolean**
Description: This parameter is ignored in GetURLImage.

Optional Parameter: **port (Default=80)**
Type: **integer**
Description: This is the port number on the server to connect with (80 = standard web server).

Optional Parameter: **filename**
Type: **string**
Description: This is the file that you would like the downloaded image to be stored in. If the filename is not fully qualified, then the HomeSeer path will be prepended to the string provided. This is recommended for VBS scripts to prevent trying to work with the byte array return which cannot be written to a file easily using VBS script accessible objects.

Returns

Return value: **page image**
Type: **byte array (.NET Object or VBScript Variant)**
Description: This is the contents of the requested web page image. If an error occurs, the text "ERROR:" will be returned followed by a reason for the error.

See Also

GetURL
GetURLEx
GetURLIE
GetURLImageEx
URLAction
SetRemoteTimeout

GetURLImageEx

Purpose

This function returns a web page image file and saves it in the file specified. This is useful for retrieving pages like weather satellite maps, and then displaying the maps in a HomeSeer device.

Parameters

Parameter: **host**
Type: **string**
Description: This is the name or IP address of host to connect to, such as "HomeSeer.com".

Parameter: **page**
Type: **string**
Description: This is the image to retrieve from the server, such as "/logo.gif". It should be fully qualified as referenced from the host parameter above, such as "\images\something\other\logo.gif" if necessary.

Parameter: **filename**
Type: **string**
Description: This is the file that you would like the downloaded image to be stored in. If the filename is not fully qualified, then the HomeSeer path will be prepended to the string provided.

Optional Parameter: **port (Default=80)**
Type: **integer**
Description: This is the port number on the server to connect with (80 = standard web server).

Returns

Return value: **result**
Type: **string**
Description: This is the result of the operation - if empty, the operation was successful. If the return value is not empty, then it will be an error message.

See Also

GetURL
GetURLEx
GetURLIE
GetURLImage
URLAction
SetRemoteTimeout

URLAction

Purpose

This command provides access to HomeSeer's internal Internet control. With it, you can post data to a web server, or get the headers from a web page.

Following are a few simple examples of the various "actions" that can be performed with this command:

POST
(Posts data to the server)

```
dim s
const server_url = "http://someserver.com/datapost/hereitis.html"
const headers="Content-Type: application/x-www-form-urlencoded"

s = hs.URLAction(server_url, "POST", data, headers)
```

GET
(Retrieves a web page - see GetURL)

```
dim s
dim data
const website = "http://www.google.com/search?sourceid=navclient&ie=UTF-8&oe=UTF-8&q=homeseer"
```

```
s = hs.URLAction(website, "GET", "", "")
```

HEAD
(Retrieves web page headers)

```
dim s

s = hs.URLAction("http://someserver.com/data/homepage.htm", "HEAD", "", "")
```

PUT
(Replaces [puts] a file at the URL)

```
dim s
dim sPage
dim sHead

sPage = (routine to read a file into the variable sPage)

s = hs.URLAction("http://someserver.com/putithere/putit.htm", "PUT", sPage, "")
```

Parameters

Parameter: **url**
Type: **string**
Description: This is the URL to post to.

Parameter: **action**
Type: **string**
Description: This is the action for the `URLAction` command, which is one of POST, PUT, HEAD, or GET.

Parameter: **data**
Type: **string**
Description: This is the URL data parameters.

Parameter: **headers**
Type: **string**
Description: This is the web page headers.

Returns

Return value: **web page**
Type: **string**
Description: This is the returned web page from the `URLAction` command, if any.

See Also

GetURL
GetURLEx
GetURLIE
GetURLImage
GetURLImageEx
SetRemoteTimeout

SetRemoteTimeout

Purpose

This function sets the number of seconds to wait for a remote host to respond when using hs.GetURL, hs.GetURLIE, or hs.ftp.

- If this function is never called, the remote timeout is set to 60 seconds.

Parameters

Parameter: **timeout**
Type: **integer**
Description: This is the number of seconds to wait.

Returns

None.

Example

```
' set the remote timeout to 30 seconds

hs.SetRemoteTimeout 30
```

See Also

GetURL
GetURLEx
GetURLIE
GetURLImage
GetURLImageEx
URLAction

GenCookieString

Purpose

This function returns a properly formatted cookie string given the parameters supplied to the function. The cookie string can then be appended to the head section of an HTML page being generated for the browser to take the appropriate cookie action. The string is created as a META set-cookie tag.

Parameters

Parameter: **name**
Type: **string**
Description: This is the name of the cookie. If commas, whitespace, semi-colons or other non-HTML friendly characters are used, then it should be noted that the name string is URL encoded in the returned string, so the name used to retrieve the cookie may have to be updated to use the URL encoded version.

Parameter: **value**
Type: **string**
Description: This is the value of the cookie. If commas, whitespace, semi-colons or other non-HTML friendly characters are used, then it should be noted that the value string is URL encoded in the returned string, so the value returned when you read the cookie back may have to be URL decoded. (See System.Web.HTTPServerUtility.URLDecode)

Parameter: **expire (optional)**
Type: **string**
Description: This is the expiration date and time for the cookie. If this parameter is omitted, then no expiration will be provided and the cookie will be erased when the web browser session ends. The date can be in any string format which can be converted by DateTime.Parse such as "April 1, 2006 11:27 PM". You may also use your system's local date/time format converted to a string value. The date provided is converted to GMT for purposes of formatting according to RFCs 822, 850, 1036 and 1123.

Parameter: **path (optional)**
Type: **string**
Description: This is the path that the cookie is valid for under the server host. If a cookie has already passed domain matching, then the pathname component of the URL is compared with the path attribute, and if there is a match, the cookie is considered valid and is sent along with the URL request. The path "/foo" would match "/foobar" and "/foo/bar.html". The path "/" is the most general path and is the default value if this parameter is omitted.

Returns

Return value: **cookie contents**
Type: **string**
Description: This is the contents of the requested cookie.

Example

This script:

```
Sub Main(parm as object)
  Dim sCookie As String = ""
  sCookie = hs.GenCookieString("Test", "MyValue", Now.AddDays(10).ToString, "/")
  hs.WriteLog("Cookie", sCookie)
End Sub
```

Generates this result:

9/21/2006 7:01:50 PM - Cookie - <meta http-equiv="Set-Cookie" content="Test=MyValue; expires=Sun, 01-Oct-2006 23:01:50 GMT; path=/;">

See Also

FTP
GetURL

Phone

In This Section

Scripting_Phone_LINEClearDTMF
Scripting_Phone_WaitMS
Scripting_Phone_StopListening
Scripting_Phone_StartListening
Scripting_Phone_Speak
Scripting_Phone_SetSpeaker
Scripting_Phone_RestoreSettings
Scripting_Phone_MBSort
Scripting_Phone_MBSave
Scripting_Phone_MBNextUnreadMessage
Scripting_Phone_MBNextReadMessage
Scripting_Phone_MBNew
Scripting_Phone_MBMessageTime
Scripting_Phone_MBMessageName
Scripting_Phone_MBMessageLength
Scripting_Phone_MBMessageFrom
Scripting_Phone_MBMessageDate
Scripting_Phone_MBMarkUnRead
Scripting_Phone_MBMarkRead
Scripting_Phone_MBGetLoggedIn
Scripting_Phone_MBGetDefault
Scripting_Phone_MBGetByName
Scripting_Phone_MBGet
Scripting_Phone_MBFirstUnreadMessage
Scripting_Phone_MBFirstReadMessage
Scripting_Phone_MBDeleteMessage
Scripting_Phone_MBCount
Scripting_Phone_MBCancelPendingNotifications
Scripting_Phone_MBAnswerMode
Scripting_Phone_MailboxClass
Scripting_Phone_LINEStopSpeaking
Scripting_Phone_LINEStatus
Scripting_Phone_LINESetVoice
Scripting_Phone_LINESetRingsCurrent
Scripting_Phone_LINESetSpeakingSpeed
Scripting_Phone_LINESetRings
Scripting_Phone_LINESetGreeting
Scripting_Phone_LINESetCIDNumber
Scripting_Phone_LINESetCIDName
Scripting_Phone_LINESetCIDInfo
Scripting_Phone_LINESetAnswerMode
Scripting_Phone_LINESendTones
Scripting_Phone_LINESendAT
Scripting_Phone_LINEScriptHasControl
Scripting_Phone_LINERingCount
Scripting_Phone_LINEResetCallTimeout
Scripting_Phone_LINEReset
Scripting_Phone_LINERecordStop
Scripting_Phone_LINERecordStart
Scripting_Phone_LINEMuteRings
Scripting_Phone_LINEIsSpeaking
Scripting_Phone_LINEHangup
Scripting_Phone_LINEGetVoice
Scripting_Phone_LINEGetDTMFString
Scripting_Phone_LINEGetDTMFCount
Scripting_Phone_LINEEnableSpeakerPhone
Scripting_Phone_LINEDisableSpeakerPhone
Scripting_Phone_LINECount
Scripting_Phone_LINEDial
Scripting_Phone_LINEAnswerSpeakerPhone
Scripting_Phone_LINEAnswerLocal
Scripting_Phone_LINEAnswer
Scripting_Phone_LastVoiceMailInfo
Scripting_Phone_LastCallerInfo
Scripting_Phone_HIPSetCallWaitingLED
Scripting_Phone_HIPSendLocalCID
Scripting_Phone_HIPCmd
Scripting_Phone_HandsetOnHook
Scripting_Phone_GetLastVoiceCommand
Scripting_Phone_CreateMessageFilename
Scripting_Phone_ContactClass
Scripting_Phone_ClearLastVoiceCommand
Scripting_Phone_CIDNumber
Scripting_Phone_CIDName
Scripting_Phone_ADRSave
Scripting_Phone_ADRNew
Scripting_Phone_ADRGet
Scripting_Phone_ADRDelete
Scripting_Phone_ADRCount

See Also

Scripting_Phone_LINEClearDTMF

LINEClearDTMF

Purpose

This function clears both the DTMF counter and the associated buffer.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to clear.

Returns

None.

See Also

Scripting_Phone_WaitMS

WaitMS

Purpose

This function waits the number of specified milliseconds. The application still processes events, but will sleep so the script does not use all the CPU.

Parameters

Parameter: **Millisecs**
Type: **Integer**
Description: The number of milliseconds to wait.

Returns

None.

Examples

' wait 2 seconds

```
hsp.WaitMS 2000
```

See Also

Scripting_Phone_StopListening

StopListening

Purpose

This function disabled the recognition engine the current line. No voice recognition will take place. A call must be in progress.

If you would like your script to work over a microphone as well as over the phone, use the system version of this command.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to access.

Returns

None.

See Also

Scripting_Phone_StartListening

StartListening

Purpose

This function enables the recognition engine to start listening on the current line. A call must be in progress.

If you would like your script to work over a microphone as well as over the phone, use the system version of this command.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to access.

Returns

None.

See Also

Scripting_Phone_Speak

Speak

Purpose

This function speaks text or a WAV file over the phone line given.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to speak to.

Parameter: **Text**
Type: **String**
Description: The text to speak. This may also be the full path to a WAV file to play over the phone.

Parameter: **Wait**
Type: **Boolean**
Description: Boolean value that causes the function to not return if set to TRUE. If set to FALSE, the speaking text is queued and the function returns immediately.

Returns

None.

See Also

Using Replacement Variables

See Also

Scripting_Phone_SetSpeaker

SetSpeaker

Purpose

This function sets the speaker for voice recognition. If you train multiple users for the voice recognition, you can use this function to switch to their profile. Call this function before performing any training. Training should be done in the HomeSeer application as there is no way to train over the phone.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The number of the telephone line to set the speaker on. Different speakers may be set for each line.

Parameter: **Name**
Type: **String**
Description: The name of the user's profile to switch to.

Returns

None.

Examples

' set the speaker to "bill"

```
hsp.SetSpeaker 1,"bill"
```

See Also

Scripting_Phone_RestoreSettings

RestoreSettings

Purpose

This function restores all program settings from the `settings.ini` file. To make some settings active, you must call hsp.LINEReset so the appropriate modem driver is reset to the new settings.

Parameters

None.

Returns

None.

See Also

Scripting_Phone_MBSort

MBSort

Purpose

This function updates the mailbox status and sorts voice messages by date. Should be called at least once before calling MBGet to get the voice messages.

Parameters

None.

Returns

None.

See Also

Scripting_Phone_MBSave

MBSave

Purpose

This function saves all configured mailbox information. Useful if a script modifies any properties of a mailbox.

Parameters

None.

Returns

None.

See Also

Scripting_Phone_MBNextUnreadMessage

MBNextUnreadMessage

Purpose

This function returns the next unread message in the given mailbox. To get all the unread messages, call MBFirstUnreadMessage, then call MBNextUnreadMessage.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to access.

Parameter: **mb**
Type: **Mailbox**
Description: The mailbox to access. This is a reference to a mailbox object. Use MBGet to get a mailbox.

Returns

Return value: **File name**
Type: **String**
Description: The file name of the voice message. Note that this file name does not include the full path to the file. Voice messages are saved in the directory messages in the HomeSeer application directory. To create the full path to the file, use the GetAppPath function like:

```
path = hsp.GetAppPath + "\messages\" + message
```

See Also

Scripting_Phone_MBNextReadMessage

MBNextReadMessage

Purpose

This function returns the next read message in the given mailbox. To get all the read messages, call MBFirstReadMessage, then call MBNextReadMessage.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to access.

Parameter: **mb**
Type: **Mailbox**
Description: The mailbox to access. This is a reference to a mailbox object. Use MBGet to get a mailbox.

Returns

Return value: **File name**
Type: **String**
Description: The file name of the voice message. Note that this file name does not include the full path to the file. Voice messages are saved in the directory messages in the HomeSeer application directory. To create the full path to the file, use the GetAppPath function like:

```
path = hsp.GetAppPath & "\messages\" & message
```

See Also

Scripting_Phone_MBNew

MBNew

Purpose

This function creates a new empty mailbox.

Parameters

None.

Returns

Return value: **Reference**
Type: **Mailbox**
Description: A reference to a new mailbox

Examples

```
dim mb

set mb = MBNew

mb.number = "555-1212"
mb.unsername = "Dad"
```

See Also

Scripting_Phone_MBMessageTime

MBMessageTime

Purpose

This function returns a string representing the time the given message was left. The time is encoded in the file name of a voice message and this function extracts the time information.

Parameters

Parameter: **Message**
Type: **String**
Description: The file name of the voice message.

Returns

Return value: **Time**
Type: **String**

See Also

Scripting_Phone_MBMessageName

MBMessageName

Purpose

This function returns the name of the user who sent the voice message. This field is the actual name returned by Caller ID. This information is encoded in the file name of the voice message and this function simply extracts it. If the Caller ID information does not include the name, this field will return the name of the caller if there is a match in the address book for the Caller ID phone number.

Parameters

Parameter: **Message**
Type: **String**
Description: The file name of the voice message.

Returns

Return value: **Caller**
Type: **String**
Description: The person who left the message. The actual name of the caller as provided by your phone company. Note that you may need to subscribe to the Caller ID name service before any value will be visible here.

See Also

Scripting_Phone_MBMessageLength

MBMessageLength

Purpose

This function returns the length of a voicemail message in seconds.

Parameters

Parameter: **Message**
Type: **String**
Description: The file name of the voice message.

Returns

Return value: **Length**
Type: **Integer**
Description: The length of the voice message, in seconds.

See Also

Scripting_Phone_MBMessageFrom

MBMessageFrom

Purpose

This function returns the name of the user who sent the voice message. The user will be either a Caller ID name or phone number. This information is encoded in the file name of the voice message and this function simply extracts it.

Parameters

Parameter: **Message**
Type: **String**
Description: The file name of the voice message.

Returns

Return value: **Caller**
Type: **String**
Description: The person who left the message. Either a name or phone number.

See Also

Scripting_Phone_MBMessageDate

MBMessageDate

Purpose

This function returns a string representing the date the given message was left. The date is encoded in the file name of a voice message and this function extracts the date information.

Parameters

Parameter: **Message**
Type: **String**
Description: The file name of the voice message.

Returns

None.

See Also

Scripting_Phone_MBMarkUnRead

MBMarkUnRead

Purpose

This function marks the voice message as not read.

Parameters

Parameter: **Message**
Type: **String**
Description: The file name of the voice message.

Returns

None.

See Also

Scripting_Phone_MBMarkRead

MBMarkRead

Purpose

This function marks the voice message as read.

Parameters

Parameter: **Message**
Type: **String**
Description: The file name of the message to mark.

Returns

None.

See Also

Scripting_Phone_MBGetLoggedIn

MBGetLoggedIn

Purpose

This function returns the mailbox index of the mailbox the caller is currently logged into. If the caller has not logged into a mailbox, this function returns 0. A caller logs into a mailbox using their passcode.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to access.

Returns

Return value: **Index**
Type: **Integer**
Description: The index of the mailbox the caller is logged into.

See Also

Scripting_Phone_MBGetDefault

MBGetDefault

Purpose

This function returns a reference to the default mailbox. The default mailbox is a specially marked mailbox that is used when the system is set up to single mailbox mode. All voice mail is left in this mailbox.

Parameters

None.

Returns

Return value: **Mailbox Class**
Type: **Object as MailboxClass**
Description: A reference to the default mailbox of class mailbox.

Examples

The following example gets the default mailbox and then accesses each voice mail message.

```
sub main()
      dim mb
      dim messages
      dim mfile
      dim count
      dim i
      set mb=hsp.MBGetDefault       ' get the default mailbox
      set messages = mb.messages    ' get the collection of messages
      count = messages.count        ' get the total number of messages
      msgbox cstr(count)
      for i=1 to count
         set mfile = messages(i)    ' get a reference to a message of type message_file
         msgbox mfile.filename      ' get the filename of the voice message
      next
end sub
```

See Also

Scripting_Phone_MBGetByName

MBGetByName

Purpose

This function returns a reference to a mailbox class using the name of the mailbox.

Parameters

Parameter: **Username**
Type: **String**
Description: The user name string of the mailbox to retrieve. The name is not case- sensitive.

Returns

Return value: **Mailbox Class**
Type: **Object as MailboxClass**
Description: A reference to a mailbox class.

See Also

Scripting_Phone_MBGet

MBGet

Purpose

This function returns a reference to a mailbox class.

Parameters

Parameter: **Index**
Type: **Integer**
Description: The index number of the mailbox to retrieve.

Returns

Return value: **Mailbox Class**
Type: **object as MailboxClass**
Description: A reference to a mailbox class.

See Also

Scripting_Phone_MBFirstUnreadMessage

MBFirstUnreadMessage

Purpose

This function returns the first unread message in the given mailbox. To get all the unread messages, this function should be called first, then call MBNextUnreadMessage.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to access.

Parameter: **mb**
Type: **Mailbox**
Description: The mailbox to access. This is a reference to a mailbox object. Use MBGet to get a mailbox.

Returns

Return value: **File name**
Type: **String**
Description: The file name of the voice message. Note that this file name does not include the full path to the file. Voice messages are saved in the directory messages in the HomeSeer application directory. To create the full path to the file, use the GetAppPath function like:

```
path = hsp.GetAppPath & "\messages\" & message
```

See Also

Scripting_Phone_MBFirstReadMessage

MBFirstReadMessage

Purpose

This function returns the first read message in the given mailbox. To get all the read messages, this function should be called first, then call MBNextReadMessage.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to access.

Parameter: **mb**
Type: **Mailbox**
Description: The mailbox to access. This is a reference to a mailbox object. Use hsp.MBGet to get a mailbox.

Returns

Return value: **File name**
Type: **String**
Description: The filename of the voice message. Note that this filename does not include the full path to the file. Voice messages are saved in the directory messages in the HomeSeer application directory. To create the full path to the file, use the GetAppPath function like:

```
path = hsp.GetAppPath & "\messages\" & message
```

See Also

Scripting_Phone_MBDeleteMessage

MBDeleteMessage

Purpose

This function deletes the given voice message. The message file is deleted.

Parameters

Parameter: **Message**
Type: **String**
Description: The file name of the voice message. The file name must not include the path.

Returns

None.

See Also

Scripting_Phone_MBCount

MBCount

Purpose

This function returns the total number of mailboxes configured. This function can be used to iterate through all the configured mailboxes.

Parameters

None.

Returns

Return value: **Number**
Type: **Integer**
Description: The total number of mailboxes configured in the application.

See Also

Scripting_Phone_MBCancelPendingNotifications

MBCancelPendingNotifications

Purpose

This function will cancel all pending notifications such as cell phone notifications (dialing out to notify someone that a message is in their mailbox), E-mail notifications, and pager notifications.

Parameters

None.

Returns

None.

See Also

Scripting_Phone_MBAnswerMode

MBAnswerMode

Purpose

This function either sets or gets the current answer mode of the system.

The answer mode is one of:

1 = multiple mailbox mode (the caller must enter a mailbox where they wish to leave a message)

2 = single mailbox mode (the caller simply leaves a message in the default mailbox)

Parameters

Parameter: **Mode** (for set)
Type: **Integer**
Description: The mode to set, either 1 or 2.

Returns

Return value: **Mode**
Type: **Integer**
Description: The current operating mode, either 1 or 2.

Examples

hsp.MBAnswerMode = modemode
return = hsp.MBAnswerMode

To set the operating mode to a single mailbox:

```
sub main()

      hsp.MBAnswerMode = 2

end sub
```

See Also

Scripting_Phone_MailboxClass

The Mailbox Class

MailboxClass Object

Various properties of a mailbox may be set and retrieved. The properties of a mailbox are defined as follows:

Keypad	Command Action
cellphone_number	The cell phone number to call with new voice messages.
email_forward	The greeting to be played to the caller entering this mailbox.
greeting	(from any level) Exit back to main menu.
notify_hi_water	The number of voice messages that must be left before a page or callback is executed. E-mail notifications are not subject to this value.
number	The mailbox number.
pager_number	The phone number of the user's pager.
passcode	A string of DTMF digits that is the passcode for this user.
tag	Holds user-defined information.
total_messages	A count of the total number of voice messages in this mailbox.
unread_messages	A count of the total number of voice messages unread in this mailbox.
username	The owner of the mailbox.

attributes	Bits defined:
	`MB_ALLOW_MESSAGES = 2 ' callers can leave messages in this mailbox`
	`MB_ALLOW_HS_VOICE_COMMANDS = 4 ' callers can access voice commands (if # enabled)`
	`MB_DEFAULT = 8 ' default, cannot delete`
	`MB_FWD_EMAIL = &H10 ' forward messages to given E-mail address`
	`MB_NOTIFY_PAGE = &H20 ' notify to pager number`
	`MB_FWD_CELLPHONE = &H40 ' forward messages to cell phone`
	`MB_NOTIFY_NO_VOICE = &H80 ' do not include the voice file in notifications`
	`MB_ATTACH_CID = &H100 ' include Caller ID number in notification`

See Also

Scripting_Phone_LINEStopSpeaking

LINEStopSpeaking

Purpose

This function stops the speaking of text-to-speech or the playing of a WAV file on the given line.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to stop speaking/playing on.

Returns

None.

See Also

Scripting_Phone_LINEStatus

LINEStatus

Purpose

This function returns the status of a call. This call can be used in a script to determine if the call is ended. If the status is `LINE_IDLE`, there is no call, and the script should exit immediately.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to access.

Returns

Return value: **Code**
Type: **String**
Description: One of the following codes:

```
LINE_IDLE      = 0    ' waiting for call
LINE_OFFERING  = 1    ' incoming call before first ring
LINE_RINGING   = 2    ' incoming call
LINE_CONNECTED = 3    ' line is active and connected to remote party
LINE_INACTIVE  = 4    ' not waiting for call, maybe no modem selected on line
LINE_BUSY      = 5    ' line busy
LINE_INUSE     = 6    ' line is in use
LINE_TIMEOUT   = 7    ' for calling, no answer
LINE_ERROR     = 8    ' line error event
LINE_DIALING   = 9    ' dialing out in progress
LINE_REORDER   = 10   ' fast busy, Hi-Phone only
```

See Also

Scripting_Phone_LINESetVoice

LINESetVoice

Purpose

This function sets a new text-to-speech voice for the given line.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to retrieve the voice from.

Parameter: **Voice**
Type: **String**
Description: The name of the new voice to set. Only SAPI5-compatible voices are supported.

Returns

None.

See Also

Scripting_Phone_LINESetRingsCurrent

LINESetRingsCurrent

Purpose

This function sets the number of rings to answer the current ringing call. Note that this call can only be made while the line is currently ringing. It may be used after Caller ID information has been examined and it has been determined that the call should be answered in a different number of rings than the default. Calling this function when the line is not ringing has no affect.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to access.

Parameter: **Rings**
Type: **Integer**
Description: The number of rings this call will answer in.

Returns

None.

See Also

Scripting_Phone_LINESetSpeakingSpeed

LINESetSpeakingSpeed

Purpose

This function sets the speaking speed for text-to-speech on the given line.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to set the speed to.

Parameter: **Speed**
Type: **Integer**
Description: The rate to set the speech to.

Returns

None.

See Also

Scripting_Phone_LINESetRings

LINESetRings

Purpose

This function sets the number of rings to answer the call. This is the same as setting the number of rings in the modem tab in the options. This is useful for setting a "Do not disturb" mode where you want to dump all callers to the voice system. Set the number of rings to 2 so that you can gather Caller ID information before answering.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to access.

Parameter: **Rings**
Type: **Integer**
Description: The number of rings to set.

Returns

None.

Examples

Set the number of rings to answer to 4 on line 1:

```
hsp.LINESetRings 1,4
```

See Also

Scripting_Phone_LINESetGreeting

LINESetGreeting

Purpose

This function sets the default greeting for the given phone line. There are two default greetings, one for a specific time range and the other for all other times. This function sets both greetings to the same phrase. This is useful if you want to set a different greeting throughout the day. In HomeSeer, you can create an event that will set the greeting for you. See the example below.

Parameters

Parameter: **Greeting**
Type: **String**
Description: The phrase to set the greeting to.

Returns

None.

Examples

To have HomeSeer set a greeting at a specific time, create an event that is triggered by the desired time, such as 8:00 AM. Then enter the following script command in the script run box on the scripting tab for the event:

```
&hsp.LINESetGreeting "Good morning, please leave a message at the beep"
```

See Also

Scripting_Phone_LINESetCIDNumber

LINESetCIDNumber

Purpose

This function sets the Caller ID number parameter to the given number. Useful if Caller ID information is gathered from some other device. This function would need to be called before the second ring, as the application handles the Caller ID information after the second ring is detected.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to answer.

Parameter: **Number**
Type: **String**
Description: The phone number of the caller.

Returns

None.

See Also

Scripting_Phone_LINESetCIDName

LINESetCIDName

Purpose

This function sets the Caller ID name parameter to the given name. Useful if Caller ID information is gathered from some other device. This function would need to be called before the second ring, as the application handles the Caller ID information after the second ring is detected.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to answer.

Parameter: **Name**
Type: **String**
Description: The name of the caller.

Returns

None.

See Also

Scripting_Phone_LINESetCIDInfo

LINESetCIDInfo

Purpose

This function sets the Caller ID name and number parameter to the given information. Useful if the Caller ID information for name and number needs to be presented at the same time to HomeSeer Phone. Unlike LINESetCIDName and LINESetCIDNumber, which have to be called after the first ring, this command signals HomeSeer Phone that Caller ID information has been set and can cause Caller ID-based events to trigger anytime the indicated line is in the ringing state.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to answer.

Parameter: **Name**
Type: **String**
Description: The name of the caller.

Parameter: **Number**
Type: **String**
Description: The phone number of the caller.

Returns

None.

See Also

Scripting_Phone_LINESetAnswerMode

LINESetAnswerMode

Purpose

This function sets the current answer mode to one of four modes. A HomeSeer event can be used to control when the answering system is turned on and off. See the parameters for the values for mode.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to access

Parameter: **Mode**
Type: **Integer**
Description: The mode to set the answering system to. Must be one of the following:

1 = answer after the number of rings set (use LINESetRings to adjust the ring count)
2 = look for Caller ID information only and don't answer calls
3 = answer external call as internal call on first ring
4 = system is disabled

Returns

None.

Examples

Set the answering system to answer on the set number of rings on line 1.

```
sub main()

      hsp.LINESetAnswerMode 1,1

end sub
```

See Also

Scripting_Phone_LINESendTones

LINESendTones

Purpose

This function sends DTMF tones over the phone line. A call must be active.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to access.

Parameter: **Digits**
Type: **String**
Description: A string of digits. Valid values are "1234567890#*"

Parameter: **Duration**
Type: **Integer**
Description: The time in milliseconds for each tone.

- This parameter is ignored for the Hi-Phone device.

Returns

None.

See Also

Scripting_Phone_LINESendAT

LINESendAT

Purpose

This function sends a raw text string directly to the attached modem. Useful for enabling special features of the modem.

- This command is not supported if you are using the Hi-Phone device.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to access.

Parameter: **Command**
Type: **String**
Description: The command string to send to the modem. The string should be terminated with a carriage-return/linefeed pair (see the example below).

Returns

Return value: **Response**
Type: **String**
Description: The response from the modem. Normally this is "OK" or "ERROR".

Examples

The example below includes a carriage return and linefeed.

```
dim r

r = hsp.LINESendAT(1,"AT" & VBCRLF)
```

See Also

Scripting_Phone_LINEScriptHasControl

LINEScriptHasControl

Purpose

This function tells HomeSeer Phone that a script is controlling the call. If called after the first ring, HomeSeer Phone will not answer the call. Ring events will still be fired. The script must answer the call with the function hsp.LINEAnswer.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to access.

Parameter: **Mode**
Type: **Boolean**
Description: The mode to set. If TRUE, only the script can answer the call. If FALSE, HomeSeer Phone will answer the call normally. The mode is reset to FALSE before the first ring on each new call.

Returns

None.

See Also

Scripting_Phone_LINERingCount

LINERingCount

Purpose

This function returns the number of rings received on the given phone line. This function can be called after a ring event in HomeSeer to take some action after a certain number of rings have been received.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to access.

Returns

Return value: **Count**
Type: **Integer**

See Also

Scripting_Phone_LINEResetCallTimeout

LINEResetCallTimeout

Purpose

This function resets the call timeout timer for the current call. The timeout timer is used to disconnect the call in the event the caller hangs up the phone. Most voice modems cannot detect when a caller hangs up.

You may need to call this function if you perform a task that takes longer than the timeout value. The timeout is set in the Phone Setup screen.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to reset the timer on.

Returns

None.

See Also

Scripting_Phone_LINEReset

LINEReset

Purpose

This function resets the given line and disconnects any call that is in progress. This call differs from the LINEHangup call in that it forces a reset to the line even if the line was already reset.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The line to reset.

Returns

None.

Examples

```
hsp.LINEReset 1
```

See Also

Scripting_Phone_LINERecordStop

LINERecordStop

Purpose

This function stops recording from the given phone line and saves the WAV information in the given file. LINERecordStart must have been called first.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to record from.

Returns

None.

See Also

Scripting_Phone_LINERecordStart

LINERecordStart

Purpose

This function starts recording from the given phone line. A call must be in progress or the function will return an error string.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to record from.

Parameter: **Filename**
Type: **String**
Description: The file name to save the recorded WAV file to. Use the CreateMessageFilename function to create a file that can be read by the HomeSeer Phone application. The message will appear in the HomeSeer Phone message list.

Returns

Return value: **Call status**
Type: **String**
Description: Returns an empty string if the call succeeded or an error string if it failed.

See Also

Scripting_Phone_LINEMuteRings

LINEMuteRings

Purpose

This function sets the ring muting status for the given line. Ring muting is the ability to not pass the incoming ring to the phones inside your home. This feature is only available on hardware that supports it. This includes devices such as the Way2Call Hi-Phone device.

- This feature is not supported on the HomeSeer PCI Voice modem.
- After making this call, the new setting is saved. If the system is restarted, it will use the new setting.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to retrieve the voice from.

Parameter: **Mode**
Type: **Integer**
Description: The mute mode to set. Modes are:

0 = No muting.
1 = Mute only the rings before caller ID is detected. After the second ring, muting is disabled, even if Caller ID is not detected.
2 = Mutes all rings. Ring signal is never passed to internal phones.

Returns

None.

Examples

```
sub main()

     hsp.LINEMuteRings 1,0       ' disable muting
     hsp.LINEMuteRings 1,1       ' enable muting until Caller ID is detected

end sub
```

See Also

Scripting_Phone_LINEIsSpeaking

LINEIsSpeaking

Purpose

This function returns the status of the text-to-speech on the given line. This function may be used to determine if speech is in progress.

Parameters

Parameter: **Index**
Type: **Integer**
Description: The phone line to disconnect.

Returns

Return value: **Speech status**
Type: **Boolean**
Description: Returns TRUE if the text-to-speech is currently speaking a phrase or FALSE if it's not.

See Also

Scripting_Phone_LINEHangup

LINEHangup

Purpose

This function disconnects the current call and hangs up the line.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to disconnect.

Returns

None.

Examples

```
hsp.LINEHangUp 1
```

See Also

Scripting_Phone_LINEGetVoice

LINEGetVoice

Purpose

This function returns the name of the text-to-speech voice currently in use on the given line. This function may be used with hsp.LINESetVoice to temporarily set a new voice then restore it back to default.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to retrieve the voice from.

Returns

Return value: **Voice name**
Type: **String**
Description: A string that is the name of the currently selected voice.

See Also

Scripting_Phone_LINEGetDTMFString

LINEGetDTMFString

Purpose

This function returns a string that is the DTMF (touch-tone) digits received. This is a buffer, and every new touch-tone detected will be added to the buffer. To clear the buffer, call LINEClearDTMF.

The "#" and "*" keys return the characters "#" and "*", respectively.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to get the buffer from.

Returns

Return value: **DTMF digits**
Type: **String**

See Also

Scripting_Phone_LINEGetDTMFCount

LINEGetDTMFCount

Purpose

This function returns the number of DTMF keys that have been detected. Use this function to check if any keys on the phone keypad have been pressed.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to check.

Returns

Return value: **DTMF value**
Type: **Integer**
Description: The count of the number of DTMF keys that have been detected

Examples

```
dim digit_count

digit_count = hsp.LINEGetDTMFCount(1)
```

See Also

Scripting_Phone_LINEEnableSpeakerPhone

LINEEnableSpeakerPhone

Purpose

This function sends the commands to the telephone hardware to enable speakerphone operation. This command must be issued when the telephone hardware is already connected to the phone line. (e.g. After answering)

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to access.

Returns

Parameter: **Status**
Type: **Integer**
Description: The status of the call - 0 = Failed, 1 = Success.

See Also

Scripting_Phone_LINEDisableSpeakerPhone

LINEDisableSpeakerPhone

Purpose

This function disables the speakerphone operation on the telephone hardware.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to access.

Returns

Parameter: **Status**
Type: **Integer**
Description: The status of the call - 0 = Failed, 1 = Success.

See Also

Scripting_Phone_LINECount

NumLines

Purpose

This function returns a count of the number of phone lines configured in the system.

Parameters

None.

Returns

Return value: **Number**
Type: **Integer**

See Also

Scripting_Phone_LINEDial

LINEDial

Purpose

This function dials the phone number given. If the line is currently in use, the error LINE_INUSE is returned.

- Voice modems can't detect when the calling party has actually answered the phone. This function will return the LINE_CONNECTED status when the modem actually starts ringing the line. If the line is busy, the LINE_BUSY error is returned. Note that the LINE_BUSY status can only be returned if the "nowait" parameter is set to FALSE. If "nowait" is set to TRY, the function returns immediately with the LINE_CONNECTED status. This is useful for quick dialing when you don't want to check for the line being busy.

The hangup parameter is used with the HomeSeer Phone switch. If this parameter is TRUE, the line is hung up immediately after dialing is complete. This will cause the switch box to reconnect the local phone.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to access.

Parameter: **Number**
Type: **String**
Description: The phone number to dial. Note that the Windows dialing properties are used to alter the phone number. If the number given includes the area code, and the area code matches the one listed in the dialing properties, it is removed.

Parameter: **Hangup**
Type: **Boolean**
Descripton: If TRUE, causes the modem to hang up immediately after dialing the number.

Parameter: **Nowait**
Type: **Boolean**
Description: If TRUE, the call returns immediately regardless if the remote party has answered. This is useful if you are connected to a PBX system and you know the PBX answers immediately. This parameter is optional, and if omitted, the system assumes a FALSE value.

If FALSE, the system will wait up to 8 seconds for the remote party to connect. Since voice modems do not notify the system when a connection is made, the delay gives time for a connection. You may have to add more of a delay in your script.

Returns

Return value: **Status**
Type: **Long (.NET Integer)**
Description: Returns a value to indicate the status of the line:

0 = LINE_IDLE
1 = LINE_OFFERING
2 = LINE_RINGING
3 = LINE_CONNECTED
4 = LINE_INACTIVE
5 = LINE_BUSY
6 = LINE_INUSE
7 = LINE_TIMEOUT (for calling)
8 = LINE_ERROR (line error event)
9 = LINE_DIALING (dialing out)
10 = LINE_REORDER (fast busy, Hi-Phone only)

Examples

The following example dials a number and speaks over the phone.

```
sub main()

      dim r

      dim voice

      r=hsp.LINEDial(1,"555-1212",false,false)

      if r=5 then

            ' line is busy

                  hs.WriteLog "Dial Error","Line busy"
```

```
            exit sub
        end if
        hsp.waitms 1000
        hsp.speak 1,"hello on the phone, how are you today?",true
        hsp.LINEHangup 1
end sub
```

See Also

Scripting_Phone_LINEAnswerSpeakerPhone

LINEAnswerSpeakerPhone

Purpose

This function forces HomeSeer Phone to answer an external call, and it sets up the telephone interface for SpeakerPhone operation. Use hsp.HandsetOnHook to detect when the user hangs up the phone.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to access.

Returns

Parameter: **Status**
Type: **Integer**
Description: The status of the call - 0 = Failed, 1 = Success.

Examples

```
hsp.LINEAnswerSpeakerPhone 1
```

See Also

Scripting_Phone_LINEAnswerLocal

LINEAnswerLocal

Purpose

This function forces HomeSeer Phone to answer an internal call. This switches the handset and the modem in and switches out the line. If you are going to take control using a script, call `hsp.LINEScriptHasControl 1,TRUE`. This will allow you to send text-to-speech (TTS) audio to the handset. Use hsp.HandsetOnHook to detect when the user hangs up the phone.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to access.

Returns

None.

Examples

```
hsp.LINEScriptHasControl 1
hsp.LINEAnswerLocal 1
hsp.Speak 1,"hello on the handset",TRUE
hsp.LINEReset 1
```

See Also

Scripting_Phone_LINEAnswer

LINEAnswer

Purpose

This function answers a call. The line must be in the offering or ringing state before calling this function.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to answer.

Returns

Return value: **Status**
Type: **Integer**
Description: Returns a zero (0) if there's no error or a non-zero value if there is an error.

See Also

Scripting_Phone_LastVoiceMailInfo

LastVoiceMailInfo

Purpose

This function returns the information on the last voice mail message left in the system, using the format specified in the HomeSeer Phone "Last Voice Mail Message" format box in the Phone Settings screen.

Parameters

Parameter: **XML information**
Type: **Boolean**
Description: If TRUE, the output is in an XML format suitable for use with other applications.

Returns

Return value: **Voicemail information**
Type: **String**
Description: String formatted according to the format string specification or in XML format.

Examples

If your "Last Voice Mail Message" format string is this:

> On `#CallDate#` at `#CallTime#` you received a call from `#CallFrom#` at `#CallNumber#`, and a message `#Length#` in length was left in `#MailBox#` mailbox.

Your returned string would appear similar to this:

> On *Thu, Jan 1, 2005* at *1:23 PM* you received a call from *Smith, John* at *603-555-1234*, and a message *0:27* in length was left in *Joe's* mailbox.

See Also

Scripting_Phone_LastCallerInfo

LastCallerInfo

Purpose

This function returns the information on the last phone call received on a specific HomeSeer Phone line number, using the format specified in the HomeSeer Phone "Last Caller Message" format box in the Phone Settings screen.

Parameters

Parameter: **Line**
Type: **Integer (.NET Short)**
Description: This is the line number that you wish to retrieve the call info from.

Returns

Return value: **Call information**
Type: **String**
Description: String formatted according to the format string specification.

Examples

If your "Last Caller Message" format string is this:

```
#CallerIDName# called at #CallerIDTimeDate# from #CallerIDNumber#
```

Your returned string would appear similar to this:

Smith, John called at *5/24/2005 11:32:41 AM* from *603-555-1234*

See Also

Scripting_Phone_HIPSetCallWaitingLED

HIPSetCallWaitingLED

Purpose

- **This command will only work with the PRO edition of the HS2 software.**

  Set the call waiting LED on a phone connected to the phone jack on the Way2Call Hi-Phone device. If the "led_on" parameter is TRUE the led is turned on, otherwise the led is turned off.

Parameters

Parameter: **Line**
Type: **Integer**

Parameter: **led_on**
Type: **Boolean**

Returns

None.

See Also

HIPSendLocalCID
HIPCmd

See Also

Scripting_Phone_HIPSendLocalCID

HIPSendLocalCID

Purpose

- **This command will only work with the PRO edition of the HS2 software.**

  Generate caller ID information to be displayed on the phones connected to the phone jack on the Way2Call Hi-Phone device.

Parameters

Parameter: **Line**
Type: **Integer**

Parameter: **Name**
Type: **string**

Parameter: **Number**
Type: **string**

Returns

None.

See Also

HIPSetCallWaitingLED
HIPCmd

See Also

Scripting_Phone_HIPCmd

HIPCmd

Purpose

This function sends a Hi-Phone specific command to the Way2Call Hi-Phone device.

Parameters

Parameter: **Line**
Type: **Integer**

Parameter: **Code**
Type: **Long**

Parameter: **Data**
Type: **Long**

Returns

None.

See Also

HIPSendLocalCID
HIPSetCallWaitingLED

See Also

Scripting_Phone_HandsetOnHook

HandsetOnHook

Purpose

This function returns the status of the local handset. This is the handset connected to the phone jack on the modem device.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to access.

Returns

Return value: **Status of line**
Type: **Boolean**
Description: Return TRUE if the local handset is on-hook (not in use) or FALSE if the handset is off-hook (in use).

Examples

```
dim hook_stat

hook_stat = hsp.HandsetOnHook(1)
```

See Also

Scripting_Phone_GetLastVoiceCommand

GetLastVoiceCommand

Purpose

This function returns the phrase last recognized over the specified phone line.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to access.

Returns

Return value: **Phrase**
Type: **String**

See Also

Scripting_Phone_CreateMessageFilename

CreateMessageFilename

Purpose

This function creates a file name for a voice message. The voice message is required to be formatted properly so it can be read by the application. The file name encodes who the message is for, the time it was received, etc.

Parameters

Parameter: **Username**
Type: **String**
Description: The user name of the mailbox the message is to be sent to.

Parameter: **Number**
Type: **Integer**
Description: The phone number of the caller, usually from Caller ID.

Parameter: **Name**
Type: **String**
Description: The name of the caller, usually from Caller ID.

Returns

Return value: **File name**
Type: **String**
Description: A string that is the complete path to the file. This string may be passed to the LINERecordStart function to start recording a voice message.

See Also

Scripting_Phone_ContactClass

Contact Class

Contact Object

HomeSeer Phone keeps an internal address book that is loaded when the program is started. Each address book entry has numerous properties that can be read and set. The class name is *contact*, and has the following properties:

Property	Description
announcement	The announcement to play when this caller calls. Either a text-to-speech string or full path to a wav file.
announce_local_wav	Text or wave file that will be played when this caller calls. The announcement is played through HomeSeer.
answer_rings	Set to TRUE to answer this call on the answer_rings_count, else FALSE to ignore
answer_rings_count	The number of rings to answer when this caller calls
business_phone	Business phone number
business_phone_2	2nd Business phone number
cell_phone	Cell phone number
cell_phone_2	2nd Cell phone number
custom1	A custom field
custom2	A custom field
email_address	email address
email_address_2	2nd email address
email_address_3	3rd email address
FIRST	First name
LAST	Last name
company_name	Company name
home_phone	Home phone number
home_phone_2	2nd Home phone number
fax_home	Home fax phone number
fax_work	Work fax phone number
pager_phone	Pager phone number
pager_pin	Pager PIN

cid_group_category	Currently not used in HomeSeer Phone, this field can be used by scripts to control answering and voicemail functions on groups of address book entries. e.g. Enter "Family" here for all family members, and then a script can control allowing the phones to ring when a family member calls.
home_address_1 through home_address_3	Three home address fields for (typically) street address information.
home_city	Home city name
home_state_province	Home state or province
home_postal_code	Home postal code for an address
home_country	Home country name for an address
business_address_1 through business_address_3	Three business address fields for (typically) street address information.
business_city	Business city name
business_state_province	Business state or province
business_postal_code	Business postal code for an address
business_country	Business country name for an address
cid_name	Name to match in the caller ID name field when a call arrives. Some phone systems may out a special string in this field like "marketing". If this field is present and matches the caller ID name field, the address entry will match the call. This field is the "Name Matching" field on the "Phone (CID Matching)" tab in the address book.
EnableRingPattern	Supported only on the Hi-Phone device. If set to TRUE, and an incoming call matches this address book entry, the phones in the home will ring with pattern as set in the RingON, RingOFF, RingDelay properties.
RingON, RingOFF, RingDelay	Hi-Phone only. These properties specify the ring pattern when the EnableRingPattern property is set to TRUE. The times are in 1/10 of a second.

cidflags	cidflags bit definitions:
	`CT_CIDFLAGS_BLOCKED = 1 ' callers with this CID are blocked`
	`CT_CIDFLAGS_ANNOUNCE = 2 ' this ID is announced`
	`CT_CIDFLAGS_SPARE1 = 4 ' spare, not used`
	`CT_CIDFLAGS_SPEC_ANN = 8 ' play special announcement to caller`
	`CT_CIDFLAGS_POPUP = &H10 ' pop up window with CID info`
	`CT_CIDFLAGS_PRIVATE = &H20 ' entry matches private CID calls`
	`CT_CIDFLAGS_OUTOFAREA = &H40 ' entry matches out of area calls`

Misc1 through Misc6	Miscellaneous string fields for use by scripts to store information. These fields are saved in the address book file but do not appear in the user interface.
MiscNum1 through MiscNum4	Miscellaneous long integer fields for use by scripts to store information. These fields are saved in the address book file but do not appear in the user interface.

flags	flags bit definitions:
	`CT_FLAGS_VRENABLED = 1 ' address book entry is enabled as voice command`
	`CT_FLAGS_HANGUP = 2 ' hang up after speaking announcement`

See Also

Scripting_Phone_ClearLastVoiceCommand

ClearLastVoiceCommand

Purpose

This function clears out the last voice command recognized to an empty string.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to access.

Returns

None.

See Also

Scripting_Phone_CIDNumber

CIDNumber

Purpose

This function returns the Caller ID number returned from the last (or current) call. This is reset when a new call arrives. The number will only be available if you have the Caller ID service. It may also be some other string like *Private* or *Out or Area* if the call was blocked.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to retrieve the information from.

Returns

Return value: **Phone number**
Type: **String**
Description: A string that is the callers phone number, or an empty string if the information was not available.

Examples

```
' get the Caller ID number

dim cnumber

cnumber = hsp.CIDNumber(1)
```

See Also

Scripting_Phone_CIDName

CIDName

Purpose

This function returns the Caller ID name returned from the last (or current) call. This is reset when a new call arrives. The name will only be available if your Caller ID service supplies names.

Parameters

Parameter: **Line**
Type: **Integer**
Description: The phone line to retrieve the information from.

Returns

Return value: **Name**
Type: **String**
Description: The caller's name or an empty string if the information was not available.

Examples

```
' get the Caller ID name

dim cname

cname = hsp.CIDName(1)
```

See Also

Scripting_Phone_ADRSave

ADRSave

Purpose

This function saves all configured address book information. This is useful if a script modifies any properties of an address book entry.

Parameters

None.

Returns

None.

See Also

Scripting_Phone_ADRNew

ADRNew

Purpose

This function returns a reference to a new address book entry (contact).

Parameters

None.

Returns

Return value: **Contact class**
Type: **Object**

See Also

Scripting_Phone_ADRGet

ADRGet

Purpose

This function returns the reference to an address book entry (contact).

Parameters

Parameter: **Index**
Type: **Integer**
Description: The index number of the address book entry to get.

Returns

Return value: **Contact class**
Type: **Object**

See Also

Scripting_Phone_ADRDelete

ADRDelete

Purpose

This function deletes an address book entry given its index. Retrieve the proper index by calling hsp.ADRCount then hsp.ADRGet to search for the proper index.

- You can't delete the private and out-of-area address book entries.

Parameters

Parameter: **Index**
Type: **Integer**
Description: The index number of the address book entry to delete.

Returns

None.

See Also

Scripting_Phone_ADRCount

ADRCount

Purpose

This function returns the number of entries that are in the address book.

Parameters

None.

Returns

Return value: **Address book entries**
Type: **Integer**

See Also

Scripts

In This Section

See Also

GetScriptPath

Purpose

This function returns the path to the directory that the last script was run from.

Parameters

None.

Returns

Return value: **path**
Type: **string**

Example

```
hs.WriteLog "Script Path", hs.GetScriptPath

Writes this (example) to the log:

      4/1/2004 2:00:00 PM~!~Script Path~!~C:\Program Files\HomeSeer\scripts\

Or for a script run from the scripts\Includes directory:

      4/1/2004 2:00:00 PM~!~Script Path~!~C:\Program Files\HomeSeer\scripts\Includes
```

See Also

IsScriptRunning

Purpose

This function indicates if a specified script is currently running.

Parameters

Parameter: **script name**
Type: **string**
Description: This is the name of the script to check.

Returns

Return value: **status**
Type: **boolean**
Description: This returns TRUE if the specified script is currently running and FALSE if it doesn't.

Example

```
' check if the script "weather.txt" is running

if hs.IsScriptRunning("weather.txt") then
  hs.writelog "info","The weather script is still running"
end if
```

See Also

GetScriptPath
RunScript
RunScriptFunc
ScriptsRunning
WaitEvents
WaitSecs

RunScript

Public Function RunScript(ByVal scr As String, ByVal Wait As Boolean, ByVal SingleInstance As Boolean) As Object

Purpose

This function runs another script. This will also return a value from the called script provided the "Main" procedure is a function.

- Scripts must be located in the scripts directory in the HomeSeer application directory (`C:\Program Files\HomeSeer 2\Scripts` by default).

Parameters

Parameter: **Script**
Type: **String**
Description: This is the file name of the script to run. Do not include the path in the script name. The "Main" procedure in the script will be run. If you need to run a specific procedure other than Main, see RunScriptFunc .

Optional Parameter: **Wait**
Type: **Boolean**
Description: When set to TRUE, the script that is calling hs.RunScript will not continue processing commands until the script referenced here is finished. Set this to False to allow the script using hs.RunScript to continue processing commands after launching the additional script.

Optional Parameter: **SingleInstance**
Type: **Boolean**
Description: When set to TRUE, only one instance of the script referenced by hs.RunScript can be running at a time, so if there is one instance already running, calling this again will result in an empty/null return and an error message written to the log.

Returns

Return value: **Value**
Type: **Object**
Description: This returns any value that the called script returns from the Main function - Sub Main will not return any values.

See Also

GetScriptPath
IsScriptRunning
RunScriptFunc
ScriptsRunning
WaitEvents
WaitSecs

RunScriptFunc

```
Public Function RunScriptFunc(ByVal Script As String, ByVal Proc As String, _
                              ByVal Params As Object, ByVal Wait As Boolean, _
                              ByVal SingleInstance As Boolean) As Object
```

Purpose

This procedure runs another script and specifies a procedure to run in that script and optional parameters. This will also return a value from the called script.

- Scripts must be located in the scripts directory in the HomeSeer application directory (`C:\Program Files\HomeSeer 3\Scripts` by default).

Parameters

Parameter: **Script**
Type: **String**
Description: This is the file name of the script to run. Do not include the path in the script name.

Parameter: **Proc**
Type: **String**
Description: This is the name of the procedure (Sub or Function) to execute. If this is left blank, the procedure "Main" will be run.

Parameter: **Params**
Type: **Object**
Description: This is a parameter or a set of parameters to send to the procedure. This can be a string or numeric value, or even an array of different values.

Optional Parameter: **Wait**
Type: **Boolean**
Description: When set to TRUE, the script that is calling hs.RunScriptFunc will not continue processing commands until the script referenced here is finished. Set this to False to allow the script using hs.RunScriptFunc to continue as soon as the other script is launched.

Optional Parameter: **SingleInstance**
Type: **Boolean**
Description: When set to TRUE, only one instance of the script referenced by hs.RunScriptFunc can be running at a time, so if there is one instance already running, calling this again will result in an empty/null return and an error message written to the log.

Returns

Return value: **Value**
Type: **Object**
Description: This returns any value (numeric, string, object) that the called script returns if the called procedure is a function.

See Also

GetScriptPath
IsScriptRunning
RunScript
ScriptsRunning
WaitEvents
WaitSecs

ScriptsRunning

Purpose

This returns a comma separated list of all of the scripts currently running in the system.

Parameters

None.

Returns

Return value: **script list**
Type: **string**
Description: This returns all of the currently running script names, separated by commas.

See Also

GetScriptPath
IsScriptRunning
RunScript
RunScriptFunc
WaitEvents
WaitSecs

WaitEvents

Purpose

This function will suspend operation of the script and allow the HomeSeer application to run. This is useful if you are waiting for a voice command or some other action that HomeSeer needs to recognize. If this function is not called, a script will time out in 30 seconds and prompt the user to either wait longer or kill the script. If this function is called within the 30 seconds, the script will not time out.

Parameters

None.

Returns

None.

Example

```
Sub Main(ByVal Parms As Object)

    Dim V As Double
    Do
       V = hs.DeviceValueEx(1234)
        If V = 41.66 Then Exit Do
       hs.WaitEvents()
       hs.WaitSecs(2)
    Loop
   hs.WriteLog("My Device", "The device has reached the proper value.")

End Sub
```

See Also

GetScriptPath
IsScriptRunning
RunScript
RunScriptFunc
ScriptsRunning
WaitSecs

WaitSecs

Purpose

This function waits a number of seconds. This will also allow other operations to take place in HomeSeer by giving up the CPU. It will also keep a script from timing out. The function will not return until the number of seconds have elapsed.

Parameters

Parameter: **seconds**
Type: **integer**
Description: This is the number of seconds to wait.

Returns

None.

Example

```
Sub Main(ByVal Parms As Object)

    Dim V As Double
    Do
       V = hs.DeviceValueEx(1234)
        If V = 41.66 Then Exit Do
       hs.WaitEvents()
       hs.WaitSecs(2)
    Loop
   hs.WriteLog("My Device", "The device has reached the proper value.")

End Sub
```

See Also

GetScriptPath
IsScriptRunning
RunScript
RunScriptFunc
ScriptsRunning
WaitEvents

Speech Recognition

In This Section

Modifying Voice Recognition Commands
Getting Last Voice Command Information
Controlling Speaker Clients

See Also

About Scripts
Applications and Plugins
Computer
Devices
Email
Events
Internet
Phone
Scripts
Strings, Global Variables, and Encryption
Time and Calendar
Text-To-Speech and Media

Modifying Voice Recognition Commands

In This Section

AddVoiceCommand
ClearAllVoiceCommands

See Also

Getting Last Voice Command Information
Controlling Speaker Clients

AddVoiceCommand

Purpose

This function will add the specified voice command to a new private command list. HomeSeer voice commands are disabled and the computer will only

listen for the commands given using this function. When the script is exited, the computer will go back to listening for regular HomeSeer voice commands.

If the script is triggered by a voice command from HomeSeer Phone, make sure you add a system call to clear all voice commands. This will tell HomeSeer Phone to restore the main menu voice commands. The statement is:

```
system.ClearAllVoiceCommands
```

Parameters

Parameter: **cmd**
Type: **string**
Description: This is the voice command to add.

Parameter: **host** (optional)
Type: **string**
Description: Leaving this a null string will apply the command to the first instance HomeSeer finds, otherwise use the hostname of the computer for this command. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

Return value: **voice command**
Type: **string**
Description: This is the specified voice command.

Example

The following script will read your E-mail messages.

```
    Sub Main(ByVal Parms As Object)

Dim Count As Integer
Count = hs.MailMsgCount
hs.Speak("You have " & Convert.ToString(Count) & " messages.", False, "")

' If no messages, exit.
If Count < 1 Then Exit Sub

hs.Speak("Would you like me to read your messages to you?", True, "")

' Clear out the last voice command recognized.
hs.LastVoiceCommand = ""

' Create our own private recognition list.
hs.AddVoiceCommand("Yes")
hs.AddVoiceCommand("Sure")
hs.AddVoiceCommand("Please")
hs.AddVoiceCommand("No")

Dim Resp As String = ""
Dim GotResponse As Boolean = False
Dim Start As Date = Now
Do
   Resp = hs.LastVoiceCommand
   If Not String.IsNullOrEmpty(Resp.Trim) Then
      GotResponse = True
      Exit Do
   End If
   hs.WaitEvents()
Loop Until Now.Subtract(Start).TotalSeconds > 15

If Not GotResponse Then
   hs.ClearAllVoiceCommands()
   hs.Speak("Goodbye.", False, "")
   Exit Sub
End If

If Resp.Trim.ToLower = "no" Then
   hs.ClearAllVoiceCommands()
   hs.Speak("OK, perhaps later.", False, "")
   Exit Sub
End If

For i As Integer = 0 To Count - 1
```

```
        hs.Speak("Message " & Convert.ToString(i), True, "")
        hs.Speak("Left on " & hs.MailDate(i), True, "")
        hs.Speak("The message is from,, " & hs.MailFrom(i), True, "")
        hs.Speak(" and the subject of the message is " & hs.MailSubject(i), True, "")
        hs.Speak(",, would you like me to read you the message?", True, "")

        Resp = ""
        GotResponse = False
        Start = Now
        Do
          Resp = hs.LastVoiceCommand
          If Not String.IsNullOrEmpty(Resp.Trim) Then
            GotResponse = True
            Exit Do
          End If
          hs.WaitEvents()
        Loop Until Now.Subtract(Start).TotalSeconds > 15

        If Not GotResponse Then
          hs.ClearAllVoiceCommands()
          hs.Speak("Goodbye.", False, "")
          Exit Sub
        End If

        Select Case Resp.Trim.ToLower
          Case "yes", "sure", "please"
            hs.Speak(hs.MailText(i), True, "")
            hs.WaitEvents()
        End Select

        hs.WaitSecs(2)

    Next

    hs.Speak("That was your last message.  Goodbye.", False, "")
    hs.ClearAllVoiceCommands()

End Sub
```

See Also

ClearAllVoiceCommands

ClearAllVoiceCommands

Purpose

This function clears all voice commands that were added with AddVoiceCommand.

Parameters

Parameter: **Host** (optional)
Type: **String**
Description: Leaving this a null string will apply the command to the first instance HomeSeer finds, otherwise use the hostname of the computer for this command. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

None.

See Also

AddVoiceCommand

Getting Last Voice Command Information

In This Section

GetLastVRCollection
GetLastVRInfo
LastCommandSelected
LastVoiceCommand
LastVoiceCommandHost
LastVoiceCommandInstance
LastVoiceCommandPhone
LastVoiceCommandRaw

See Also

Modifying Voice Recognition Commands
Controlling Speaker Clients

GetLastVRCollection

Purpose

This function gets the last voice command recognized by all speaker clients and HomeSeer Phone lines. The return is a simple array of clsLastVR objects.

Parameters

None.

Returns

Return value: **LastVR**
Type: **Array of clsLastVR**
Description: This returns the last voice command that HomeSeer recognized on all connected speaker clients and HomeSeer phone lines in an array of clsLastVR objects.

- Note - if a given speaker client is connected but has not been used for VR since HomeSeer was started, it will not be a part of the returned array.

See Also:

clsLastVR
GetLastVRInfo
LastCommandSelected
LastVoiceCommand
LastVoiceCommandPhone
LastVoiceCommandHost
LastVoiceCommandInstance
LastVoiceCommandRaw

See Also

GetLastVRInfo
LastCommandSelected
LastVoiceCommand
LastVoiceCommandHost
LastVoiceCommandInstance
LastVoiceCommandPhone
LastVoiceCommandRaw

clsLastVR

clsLastVR is an object class used with GetLastVRInfo and GetLastVRCollection and returns information about the last recognized voice command given to HomeSeer.

Here are the properties of the class:

Property	Type	Description
Raw	String	This is the raw voice command as it was heard and recognized by the VR engine.
Parsed	String	This is the parsed voice command. For HomeSeer generated voice commands, this string will contain special indicators for the matched device or event - it will not match the spoken text.
Host	String	This is the host name of the speaker client the recognized phrase was spoken to, or 'Phone' if it was spoken via HomeSeer Phone's local or remote interaction.
Instance	String	This is the instance name of the speaker client the recognized phrase was spoken to, or the phone line number if it was spoken via HomeSeer Phone's local or remote interaction.
VRTime	Date	This is the date/time the phrase was recognized.
ID	Integer	This is the voice recognition context ID number that was matched for the recognized phrase.

See Also:

GetLastVRInfo
GetLastVRCollection
LastCommandSelected
LastVoiceCommand
LastVoiceCommandPhone
LastVoiceCommandHost
LastVoiceCommandInstance
LastVoiceCommandRaw

See Also

GetLastVRInfo

Purpose

This function gets the last voice command recognized by a given speaker client/instance. The return is a clsLastVR object matching the speaker client host name and instance provided.

Parameters

Parameter: **host**
Type: **string**
Description: This is the host name for the speaker client to retrieve the last recognized VR information from.

Parameter: **instance**
Type: **string**
Description: This is the instance name for the speaker client to retrieve the last recognized VR information from.

- Note - if a given speaker client is connected but has not been used for VR since HomeSeer was started, it will not be returned with this command.

Returns

Return value: **LastVR**
Type: **clsLastVR**
Description: This returns the last voice command that HomeSeer recognized on the given host:instance in a clsLastVR object, or 'nothing' if no matching host:instance was found.

See Also:

clsLastVR
GetLastVRCollection
LastCommandSelected
LastVoiceCommand
LastVoiceCommandPhone
LastVoiceCommandHost
LastVoiceCommandInstance
LastVoiceCommandRaw

See Also

GetLastVRCollection
LastCommandSelected
LastVoiceCommand
LastVoiceCommandHost
LastVoiceCommandInstance
LastVoiceCommandPhone
LastVoiceCommandRaw

clsLastVR

clsLastVR is an object class used with GetLastVRInfo and GetLastVRCollection and returns information about the last recognized voice command given to HomeSeer.

Here are the properties of the class:

Property	Type	Description
Raw	String	This is the raw voice command as it was heard and recognized by the VR engine.
Parsed	String	This is the parsed voice command. For HomeSeer generated voice commands, this string will contain special indicators for the matched device or event - it will not match the spoken text.
Host	String	This is the host name of the speaker client the recognized phrase was spoken to, or 'Phone' if it was spoken via HomeSeer Phone's local or remote interaction.

Instance	String	This is the instance name of the speaker client the recognized phrase was spoken to, or the phone line number if it was spoken via HomeSeer Phone's local or remote interaction.
VRTime	Date	This is the date/time the phrase was recognized.
ID	Integer	This is the voice recognition context ID number that was matched for the recognized phrase.

See Also:

GetLastVRInfo
GetLastVRCollection
LastCommandSelected
LastVoiceCommand
LastVoiceCommandPhone
LastVoiceCommandHost
LastVoiceCommandInstance
LastVoiceCommandRaw

See Also

LastCommandSelected

Purpose

This function gets the event name of the last voice command. This works the same as the LastVoiceCommand function except it will return the actual name of the voice command. This is useful if you wanted to do some other action to the event, like delete it or disable it and you need that actual event name. This is a read-only property.

Parameters

None.

Returns

Return value: **event name**
Type: **string**
Description: This returns the name of the event that was triggered by the last voice command.

See Also:

clsLastVR
GetLastVRInfo
GetLastVRCollection
LastVoiceCommand
LastVoiceCommandPhone
LastVoiceCommandHost
LastVoiceCommandInstance
LastVoiceCommandRaw

See Also

GetLastVRCollection
GetLastVRInfo
LastVoiceCommand
LastVoiceCommandHost
LastVoiceCommandInstance
LastVoiceCommandPhone
LastVoiceCommandRaw

LastVoiceCommand

Purpose

This function gets the last voice command recognized by a speaker client. This is a read-only property. This value is the parsed (processed) voice recognition string, which means that some parts of the command may be replaced by values or codes that HomeSeer uses to interpret what was spoken. See LastVoiceCommandRaw to get the unparsed (raw) phrase.

Parameters

None.

Returns

Return value: **voice command**
Type: **string**
Description: This returns the last voice command that HomeSeer recognized. This is useful for obtaining the actual voice command when the given voice command contains many optional words.

Example

If a voice command was set to:

```
tv channel (0|1|2|3|4|5|6|7|8|9)
```

and the user spoke "tv channel 4", this function would return the string "*t*v channel 4"

Create an event name tv channel. Set the voice command to:

```
tv channel (0|1|2|3|4|5|6|7|8|9)
```

Set the actions of the event to run the following script. When you speak a phrase like "tv channel 2", a message box will pop up giving you the actual command the system recognized.

```
sub main()

    dim v
    v=hs.LastVoiceCommand
    msgbox "I heard "&v

end sub
```

See Also:

clsLastVR
GetLastVRInfo
GetLastVRCollection
LastCommandSelected
LastVoiceCommandPhone
LastVoiceCommandHost
LastVoiceCommandInstance
LastVoiceCommandRaw

GetLastVRCollection
GetLastVRInfo
LastCommandSelected
LastVoiceCommandHost
LastVoiceCommandInstance
LastVoiceCommandPhone
LastVoiceCommandRaw

LastVoiceCommandHost

Purpose

This function gets the host name of the speaker client for the last voice command recognized by a speaker client. This is a read-only property. This command will return "Phone" if the last recognized command came from the a HomeSeer Phone line.

Parameters

None.

Returns

Return value: **host name**
Type: **string**
Description: This returns the host name where the speaker client is running that the last voice command that HomeSeer recognized originated from. If the phone interface was used, this command returns the text: Phone

See Also:

clsLastVR
GetLastVRInfo
GetLastVRCollection
LastCommandSelected
LastVoiceCommand
LastVoiceCommandPhone
LastVoiceCommandInstance
LastVoiceCommandRaw

See Also

GetLastVRCollection
GetLastVRInfo
LastCommandSelected
LastVoiceCommand
LastVoiceCommandInstance
LastVoiceCommandPhone
LastVoiceCommandRaw

LastVoiceCommandInstance

Purpose

This function gets the instance name of the speaker client for the last voice command recognized by a speaker client. This is a read-only property. This command will return a phone line number (e.g. "1", "2", etc.) if the last recognized command came from a HomeSeer Phone line.

Parameters

None.

Returns

Return value: **instance name**
Type: **string**
Description: This returns the instance name where the speaker client is running that the last voice command that HomeSeer recognized originated from. If the phone interface was used, this command returns the HomeSeer Phone line number as text.

See Also:

clsLastVR
GetLastVRInfo
GetLastVRCollection
LastCommandSelected
LastVoiceCommand
LastVoiceCommandPhone
LastVoiceCommandHost
LastVoiceCommandRaw

See Also

GetLastVRCollection
GetLastVRInfo
LastCommandSelected
LastVoiceCommand
LastVoiceCommandHost
LastVoiceCommandPhone
LastVoiceCommandRaw

LastVoiceCommandPhone

Purpose

This function gets the last voice command recognized by HomeSeer Phone. This is a read-only property. This value is the parsed (processed) voice recognition string, which means that some parts of the command may be replaced by values or codes that HomeSeer uses to interpret what was spoken. See LastVoiceCommandRaw to get the unparsed (raw) phrase.

- HomeSeer Phone is required to do voice recognition over the telephone.

Parameters

None.

Returns

Return value: **voice command**
Type: **string**
Description: This returns the last voice command that HomeSeer recognized via the telephone. This is useful for obtaining the actual voice command when the given voice command contains many optional words.

Example

If a voice command was set to:

```
tv channel (0|1|2|3|4|5|6|7|8|9)
```

and the user spoke "tv channel 4", this function would return the string "*tv* channel 4"

Create an event name tv channel. Set the voice command to:

```
tv channel (0|1|2|3|4|5|6|7|8|9)
```

Set the actions of the event to run the following script. When you speak a phrase like "tv channel 2", a message box will pop up giving you the actual command the system recognized.

```
sub main()

     dim v
     v=hs.LastVoiceCommandPhone
     hs.WriteLog "LVCP", "I heard " & v & " from the phone."

end sub
```

See Also:

clsLastVR
GetLastVRInfo
GetLastVRCollection
LastCommandSelected
LastVoiceCommand
LastVoiceCommandPhone
LastVoiceCommandHost
LastVoiceCommandInstance
LastVoiceCommandRaw

See Also

GetLastVRCollection
GetLastVRInfo
LastCommandSelected
LastVoiceCommand
LastVoiceCommandHost
LastVoiceCommandInstance
LastVoiceCommandRaw

LastVoiceCommandRaw

Purpose

This function gets the last voice command recognized by a speaker client or HomeSeer phone in raw (unparsed) format. The unparsed format matches the phrase spoken by the user. This is a read-only property.

Parameters

None.

Returns

Return value: **voice command**
Type: **string**
Description: This returns the last voice command that HomeSeer recognized in unparsed form. In unparsed form, the spoken phrase "Turn on the Kitchen Light" will return the same text. In parsed form, the phrase might return something like "Turn on DV:5427"

See Also:

clsLastVR
GetLastVRInfo
GetLastVRCollection
LastCommandSelected
LastVoiceCommand
LastVoiceCommandPhone
LastVoiceCommandHost
LastVoiceCommandInstance

See Also

GetLastVRCollection
GetLastVRInfo
LastCommandSelected
LastVoiceCommand
LastVoiceCommandHost
LastVoiceCommandInstance
LastVoiceCommandPhone

Controlling Speaker Clients

In This Section

GetListenStatus
ListenMode
ListenForCommands
SetSpeaker
StartListen
StopListen

See Also

Modifying Voice Recognition Commands
Getting Last Voice Command Information

GetListenStatus

Purpose

This function returns the listening status of a specific speaker client (host or host:instance).

Parameters

Parameter: **host**
Type: **string**
Description: Leaving this a null string will return the status for the first instance HomeSeer finds, otherwise use the hostname of the computer you are interested in determining the listening status of. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

Return value: **listening status**
Type: **boolean**
Description: TRUE indicates that the speaker app instance is listening.

See Also

ListenMode
ListenForCommands
SetSpeaker
StartListen
StopListen

ListenMode

Purpose

This function indicates the current listening mode.

Parameters

None.

Returns

Return value: **mode**
Type: **integer**
Description: This returns the current listen mode which is define as:

1 = Not Listening
2 = Listening for commands
3 = Listening for attention

See Also

GetListenStatus
ListenForCommands
SetSpeaker
StartListen
StopListen

ListenForCommands

Purpose

This function will switch the computer from either listening for event name commands or listening for the attention phrase.

Parameters

Parameter: **action**
Type: **boolean**
Description: Use TRUE to listen for event name commands and FALSE to listen for the attention phrase.

Returns

None.

Example

```
sub main()

    ' listen only for attention phrase
 hs.ListenForCommands FALSE

end sub
```

See Also

GetListenStatus
ListenMode
SetSpeaker
StartListen
StopListen

SetSpeaker

Purpose

This procedure changes the speaker profile on one or more Speaker clients to the profile name provided.

Parameters

Parameter: **speaker_name**
Type: **string**
Description: This is the name of the speaker profile to switch to. The speaker profile name must match one of the available speaker profiles on the computer that the HomeSeer Speaker client is running on.

Parameter: **host** (optional)
Type: **string**
Description: Leaving this a null string will apply the command to the first instance HomeSeer finds, otherwise use the hostname of the computer for this command. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

None.

See Also

GetListenStatus
ListenMode
ListenForCommands
StartListen
StopListen

StartListen

Purpose

This function starts the voice recognition engine if it is not already started. For scripts that are to be used over the phone, use the System functions.

Parameters

Parameter: **host** (optional)
Type: **string**
Description: Leaving this a null string will apply the command to the first instance HomeSeer finds, otherwise use the hostname of the computer for this command. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

None.

See Also

GetListenStatus
ListenMode
ListenForCommands
SetSpeaker
StopListen

StopListen

Purpose

This function stops the voice recognition engine if it is not already stopped. For scripts that are to be used over the phone, use the System functions.

Parameters

Parameter: **host** (optional)
Type: **string**
Description: Leaving this a null string will apply the command to the first instance HomeSeer finds, otherwise use the hostname of the computer for this command. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

None.

See Also

GetListenStatus
ListenMode
ListenForCommands
SetSpeaker
StartListen

Strings, Global Variables, and Encryption

In This Section

See Also

About Scripts
Applications and Plugins
Computer
Devices
Email
Events
Internet
Phone
Scripts
Speech Recognition
Time and Calendar
Text-To-Speech and Media

String Utilities

In This Section

StringItem

See Also

Global Variables
Encryption

StringItem

Purpose

This function retrieves the substring from a string. Given a string like "data;data1;data2", this function will retrieve the string at the specified index. For instance, passing the index as 2 would return the string "data1".

Parameters

Parameter: **str**
Type: **string**
Description: This is the string to search in.

Parameter: **index**
Type: **long**
Description: This is the number of the string you wish returned. 1=string 1,2=string 2, etc.

Parameter: **sep**
Type: **string**
Description: This is the string that separates all other strings.

Returns

Return value: **string part**
Type: **string**
Description: This is the part of the string requested.

See Also

Global Variables

In This Section

CreateVar
DeleteVar
GetVar
SaveVar

See Also

String Utilities
Encryption

CreateVar

Purpose

This function creates a new global variable. The variable may be accessed by the functions SaveVar and GetVar. The variable is global in scope and can only be destroyed with the DeleteVar function or exiting the application.

The variable created is an object and can be used to hold any variable type, including references to objects.

Parameters

Parameter: **name**
Type: **string**
Description: This is the name of the variable.

Returns

Return value: **error code**
Type: **string**
Description: This is an empty string if there's no error. Otherwise, an error string will be returned if the variable already exists.

Example

```
dim errst

errst = CreateVar("myvar")

if errst <> "" then
      msgbox "Error creating variable"
end if
```

See Also

DeleteVar
GetVar
SaveVar

DeleteVar

Purpose

This function deletes a global variable or reference to an object that was created by CreateVar. If the variable does not exist, the function does nothing.

Parameters

Parameter: **name**
Type: **string**
Description: This is the name of the variable.

Returns

None.

See Also

CreateVar
GetVar
SaveVar

GetVar

Purpose

This function finds the variable associated with the name parameter and returns it.

Parameters

Parameter: **name**
Type: **string**
Description: This is the name of the variable.

Returns

Return value: **variable item**
Type: **variant**
Description: This returns the variable saved.

Example

```
dim myvar

myvar = hs.GetVar("myvar")

' if "myvar" is an object, then get the variable with:

set myvar = hs.GetVar("myvar")
```

See Also

CreateVar
DeleteVar
SaveVar

SaveVar

Purpose

This function saves the variable contained in the `obj` parameter. The parameter may be any variable type such as a string or integer, or it may be a reference to an object created with CreateObject.

Parameters

Parameter: **name**
Type: **string**
Description: This is the name of the variable.

Parameter: **obj**
Type: **object**
Description: This is the object to be saved.

Returns

Return value: **error code**

Type: **string**
Description: This returns an empty string if no error occurred and returns an error string if one did occur.

Example

```
dim errst
dim myvalue

myvalue = 10
errst = hs.SaveVar("myvar",myvalue)
```

See Also

CreateVar
DeleteVar
GetVar

Encryption

In This Section

EncryptString
EncryptStringEx
DecryptString

See Also

String Utilities
Global Variables

EncryptString

Purpose

This function encrypts a string using an encryption password that you specify. Although many unprintable characters can be written to a text file successfully, Windows terminates a text string with a carriage return/line-feed character combination. The `bRecurse` parameter is provided to cause the function to recursively encrypt the data until it detects no carriage return or line-feed characters in it. The string may then be written to a text file such as when you save it in an INI file using SaveINISetting. Using `bRecurse` on a large amount of text is NOT recommended as it may recursively encrypt for a long time in an attempt to remove carriage return and line-feed characters, or the function may result in an error due to too many attempts to recursively encrypt. Another solution for writing encrypted data to a text file safely is to convert it to a text representation of HEX data. See the example below.

Parameters

Parameter: **sToEncrypt**
Type: **string**
Description: This is the text that you want encrypted.

Parameter: **sPassword**
Type: **string**
Description: This is the user-created text string to encrypt the text with.

Returns

Return value: **data**
Type: **string**
Description: This returns a string containing an encrypted form of `sToEncrypt`, encrypted using `sPassword`. This string is not limited to printable

characters only, so care should be taken in the storage of this data in files.

Example

```
Sub Main(ByVal Parms As Object)

    If Parms Is Nothing Then Exit Sub

    ' Encrypt the combination to my vault full of money.  The combination
    '   that I just entered is stored in the variable sCombEntered.
    Dim sCombEntered As String = Convert.ToString(Parms)

    Dim sCombination As String = ""
    sCombination = hs.EncryptString(sCombEntered, "For Spouse Only Spend Wisely", False)

    ' I now have my encrypted combination in sCombination.  I must remember
    '   to use HomeSeer's or Microsoft's script encrypters on this script
    '   since my password string is plainly visible above!

    ' I want to store the combination in a text file, so let's Base64 encode it.
    Dim bteArray() As Byte
    bteArray = Encoding.ASCII.GetBytes(sCombination)
    Dim sOutput As String = ""
    sOutput = Convert.ToBase64String(bteArray, Base64FormattingOptions.None)

    ' Now I have sOutput as a text representation of bytes, I can write that to an INI
    '   file and reverse the process of encoding it to Base64 to unencode it.
    hs.SaveINISetting("Passwords", "Vault", sOutput, "MyPasswords.ini")

End Sub
```

See Also:

EncryptStringEx
DecryptString

See Also

EncryptStringEx
DecryptString

EncryptStringEx

Purpose

This function encrypts a string using an encryption password that you specify.

Notes

Encrypted strings using this function are encrypted using strong (AES/Rijndael) encryption - the data can NOT be recovered if the password(s) are lost. The resulting data may have unprintable characters, so you may not be able to save it using INI functions. Another solution for writing encrypted data to a text file safely is to convert it to a text representation of HEX data. See the example used in EncryptString.

Parameters

Parameter: **Text**
Type: **String**
Description: This is the text that you want encrypted.

Parameter: **Password**
Type: **String**
Description: This is the user-created text string to encrypt the text with.

Parameter: **KeyModifier**
Type: **String**
Description: This parameter may be used to provide further user-specific encryption of the data - it is a modifier used with the password parameter to create the encryption key.

Returns

Return value: **data**
Type: **string**
Description: This returns a string containing an encrypted form of `Text`, encrypted using `Password (and KeyModifier if provided)`. This string is not limited to printable characters only, so care should be taken in the storage of this data in files.

See Also:

EncryptString
DecryptString

See Also

EncryptString
DecryptString

DecryptString

Purpose

This function decrypts a string using a decryption password that you specify.

Parameters

Parameter: **sToDecrypt**
Type: **String**
Description: This is the text that you want decrypted (unencrypted).

Parameter: **sPassword**
Type: **String**
Description: This is the user-created text string to encrypt the text with.

Parameter: **KeyModifier (Optional)**
Type: **String**
Description: This optional parameter is the modifier text to use with the password to create the key - if EncryptStringEx was used to encrypt the string and a key modifier was used, you must specify the same key modifier here.

Returns

Return value: **Data**
Type: **String**
Description: This returns a string containing a decrypted form of `sToDecrypt`, decrypted using `sPassword`. Only the same value of `sPassword` used to encrypt the string will return the original string in this function.

Example

```
Sub Main(ByVal Parms As Object)

    ' Decrypt the combination to my vault full of money.

    ' First I have to read the encrypted string from a file and unencode it.
    ' The string was Base64 encoded so that it could be safely written to a text file.
    Dim sCombination As String = ""
    sCombination = hs.GetINISetting("Passwords", "Vault", "NOTHING", "MyPasswords.ini")
    If sCombination Is Nothing OrElse String.IsNullOrEmpty(sCombination.Trim) Then
        hs.WriteLog("Error", "Encrypted combination was not read from the INI file properly.")
        Exit Sub
```

```
    End If
    If sCombination.Trim.ToLower = "nothing" Then
       hs.WriteLog("Warning", "The encrypted vault password was not found in our passwords INI file.")
       Exit Sub
    End If

    ' Now decode the string back into an array of bytes.
    Dim bteArray() As Byte
    bteArray = Convert.FromBase64String(sCombination)
    Dim sCombEntered As String = ""
    sCombEntered = Encoding.ASCII.GetString(bteArray)

    ' Now we have the encrypted combination, so let's decrypt it. (We'll re-use sCombination)
    sCombination = hs.DecryptString(sCombEntered, "For Spouse Only Spend Wisely")

    ' I now have my decrypted combination in sCombination.  I must remember
    '   to use HomeSeer's or Microsoft's script encrypters on this script
    '   since my password string is plainly visible above!

End Sub
```

See Also:

EncryptString
EncryptStringEx

See Also

EncryptString
EncryptStringEx

Time and Calendar

In This Section

Time Related
Calendar Related

See Also

About Scripts
Applications and Plugins
Computer
Devices
Email
Events
Internet
Phone
Scripts
Speech Recognition
Strings, Global Variables, and Encryption
Text-To-Speech and Media

Time Related

In This Section

See Also

Calendar Related

LocalTimeZone

Purpose

This function returns an offset in minutes from UTC (Universal Time Coordinate) for your time zone.

- The offset is based upon UTC, which is the time standard used since 1972, and not GMT, which was the previous standard.

Parameters

None.

Returns

Return value: **Offset**
Type: **Integer**
Description: This returns the current time zone offset from UTC for the time zone set on your HomeSeer computer.

Example

```
hs.WriteLog "TimeZone","My timezone offset here in Eastern Daylight Time from UTC is " &
CStr(hs.LocalTimeZone / 60) & " hours."
```

Results in this being written to the log:

```
4/14/2004 3:00:00 PM~!~TimeZone~!~My timezone offset here in Eastern Daylight Time from UTC is 5
hours.
```

See Also

SolarNoon
Sunrise
SunriseDt
Sunset
SunsetDt
TimeZoneName

SolarNoon

Purpose

This function returns the time of solar noon. This is a read-only property.

Parameters

None.

Returns

Return value: **solar noon time**
Type: **date**
Description: This is a date item representing the time of solar noon, the period at which the sun appears directly overhead a location.

Example

```
sub main()

   dim t

   t=hs.SolarNoon
   msgbox "Solar Noon is at " & FormatDateTime(t, vbLongTime)

end sub
```

See Also

LocalTimeZone
Sunrise
SunriseDt
Sunset
SunsetDt
TimeZoneName

Sunrise

Purpose

This function returns the time of sunrise. This is a read-only property.

Parameters

None.

Returns

Return value: **sunrise time**
Type: **string**
Description: This is a string representing the time of sunrise. The string returned is formatted according to your system's setting for time display but with seconds removed (e.g., if there are three fields separated by colons, the third one is removed).

Example

```
sub main()

   dim t

   t=hs.Sunrise
   msgbox "Sunrise is at " & t

end sub
```

See Also

LocalTimeZone
SolarNoon
SunriseDt
Sunset
SunsetDt
TimeZoneName

SunriseDt

Purpose

This function returns the time of sunrise. This is a read-only property.

Parameters

None.

Returns

Return value: **sunrise time**
Type: **date**
Description: This is a date type representing the time of sunrise.

Example

```
sub main()

   dim t

   t=hs.SunriseDt
   msgbox "Sunrise is at " & FormatDateTime(t, vbLongTime)

end sub
```

See Also

LocalTimeZone
SolarNoon
Sunrise
Sunset
SunsetDt
TimeZoneName

Sunset

Purpose

This function returns the time of sunset. This is a read-only property.

Parameters

None.

Returns

Return value: **sunset time**
Type: **string**
Description: This is a string representing the time of sunset. The string returned is formatted according to your system's setting for time display but with seconds removed (e.g., if there are three fields separated by colons, the third one is removed).

Example

```
sub main()

   dim t

   t=hs.Sunset
   msgbox "Sunset is at " & t

end sub
```

See Also

LocalTimeZone
SolarNoon
Sunrise
SunriseDt
SunsetDt
TimeZoneName

SunsetDt

Purpose

This function returns the time of sunset. This is a read-only property.

Parameters

None.

Returns

Return value: **sunset time**
Type: **date**
Description: This is a date type representing the time of sunset.

Example

```
sub main()

   dim t

   t=hs.SunsetDt
   msgbox "Sunset is at " & FormatDateTime(t, vbLongTime)

end sub
```

See Also

LocalTimeZone
SolarNoon
Sunrise
SunriseDt
Sunset
TimeZoneName

TimeZoneName

Purpose

This function returns the name of the PC's time zone setting. This is a read-only property.

Parameters

None.

Returns

Return value: **time zone**
Type: **string**
Description: This is the name of the time zone as read from the operating system.

Example

```
sub main()

   dim t

   t=hs.TimeZoneName
   msgbox "The TimeZone is " & t

end sub
```

See Also

LocalTimeZone
SolarNoon
Sunrise
SunriseDt
Sunset
SunsetDt

Calendar Related

In This Section

See Also

Time Related

DaylightSavings

Purpose

This function returns whether daylight savings is currently active. This is a read-only property.

Parameters

None.

Returns

Return value: **Currently in daylight savings**
Type: **Boolean**
Description: The return value (TRUE or FALSE) indicates whether the current date falls under daylight savings time as reported by the operating system.

- Daylight savings is not used in all locations.

Example

```
sub main()

   if hs.DayLightSavings then
    hs.WriteLog "We are currently in daylight savings!"
   end if

end sub
```

See Also

DaysInMonth
DaysLeftInMonth
DaysLeftInYear
EvenOddMonth
EvenOddDay
GetLastWeekday
GetSpecialDay
IsSpecialDay
IsWeekday
IsWeekend
Moon
Weekdays
WeekEndDays
WeekNumber
WeekNumberEx
WeeksLeftInYear
WeeksLeftInYearEx

DaysInMonth

Purpose

This function returns the number of days in the month of a date value supplied to it.

Parameters

Parameter: **Date**
Type: **Date**
Description: This is a date object for which you wish to know how many days are in that month. The day of the month in the date object is ignored.

Returns

Return value: **number of days**
Type: **Integer**

Example

```
Dim dte As Date = DateTime.Parse("April 1, 2006")
hs.WriteLog("Info","There are " & hs.DaysInMonth(dte).ToString & " days in the month of April, 2006")
```

See Also

DaysLeftInMonth
DaysLeftInYear
WeekNumber
WeeksLeftInYear

See Also

DaylightSavings
DaysLeftInMonth
DaysLeftInYear
EvenOddMonth
EvenOddDay
GetLastWeekday
GetSpecialDay
IsSpecialDay
IsWeekday
IsWeekend
Moon
Weekdays
WeekEndDays
WeekNumber
WeekNumberEx
WeeksLeftInYear
WeeksLeftInYearEx

DaysLeftInMonth

Purpose

This function returns a value indicating how many days are left in the current month.

Parameters

None.

Returns

Return value: **Number of days**
Type: **Integer**
Description: The number of days remaining in the current month.

Example

```
hs.WriteLog("Info","There are " & hs.DaysLeftInMonth.ToString & " days left in the month.")
```

See Also

DaysInMonth
DaysLeftInYear
WeekNumber
WeeksLeftInYear

See Also

DaylightSavings
DaysInMonth
DaysLeftInYear
EvenOddMonth
EvenOddDay
GetLastWeekday
GetSpecialDay
IsSpecialDay
IsWeekday
IsWeekend
Moon
Weekdays
WeekEndDays
WeekNumber
WeekNumberEx
WeeksLeftInYear
WeeksLeftInYearEx

DaysLeftInYear

Purpose

This function returns a value indicating how many days are left in the current year.

Parameters

None.

Returns

Return value: **Number of days**
Type: **Integer**
Description: The number of days remaining in the current year.

Example

```
hs.WriteLog("Info","There are " & hs.DaysLeftInYear.ToString & " days left in the year.")
```

See Also

DaysInMonth
DaysLeftInMonth
WeekNumber
WeeksLeftInYear

See Also

EvenOddMonth

Purpose

This function returns a value indicating whether the provided day of the month is even or odd.

Parameters

Parameter: **date**
Type: **date**
Description: This is the date that you wish to check for being even or odd for the month.

Returns

Return value: **CD_DAY_EvenOdd**
Type: **Enum (Integer) 0 = Even, 1 = Odd**
Description: The return is a .NET Enum equivalent to an integer value.
The return value converted to string with the .ToString method will return the word Even or Odd, but when converted to an integer value and then to a string it will display 0 or 1.

Example

```
Dim dte As Date = DateTime.Parse("April 1, 2006")
hs.WriteLog("Info","April 1 of 2006 is an " & hs.EvenOddMonth(dte).ToString & " day of the month.")
```

See Also

EvenOddDay

See Also

DaylightSavings
DaysInMonth
DaysLeftInMonth
DaysLeftInYear
EvenOddDay
GetLastWeekday
GetSpecialDay
IsSpecialDay
IsWeekday
IsWeekend
Moon
Weekdays
WeekEndDays
WeekNumber
WeekNumberEx
WeeksLeftInYear
WeeksLeftInYearEx

Home > Scripting > Time and Calendar > Calendar Related > EvenOddDay

EvenOddDay

Purpose

This function returns a value indicating whether the provided day of the year is even or odd.
(The day of the month may be odd, but it may still be an even number for the year.)

Parameters

Parameter: **date**
Type: **date**
Description: This is the date that you wish to check for being even or odd for the year.

Returns

Return value: **CD_DAY_EvenOdd**
Type: **Enum (Integer) 0 = Even, 1 = Odd**
Description: The return is a .NET Enum equivalent to an integer value.
The return value converted to string with the .ToString method will return the word Even or Odd,
but when converted to an integer value and then to a string it will display 0 or 1.

Example

```
Dim dte As Date = DateTime.Parse("April 1, 2006")
hs.WriteLog("Info","April 1 of 2006 is an " & hs.EvenOddDay(dte).ToString & " day.")
```

See Also

EvenOddMonth

See Also

DaylightSavings
DaysInMonth
DaysLeftInMonth
DaysLeftInYear
EvenOddMonth
GetLastWeekday
GetSpecialDay
IsSpecialDay
IsWeekday
IsWeekend
Moon
Weekdays
WeekEndDays
WeekNumber
WeekNumberEx
WeeksLeftInYear
WeeksLeftInYearEx

GetLastWeekday

Purpose

This function returns a date representing the last weekday of the month from the date provided.

Parameters

Parameter: **date**
Type: **date**
Description: This is a date in the month for which you wish to know the date of the last weekday of that month.

Returns

Return value: **last weekday date**
Type: **date**
Description: The date of the last weekday of the month.

Example

```
Dim dte As Date = DateTime.Parse("April 1, 2006")
hs.WriteLog("Info","The last weekday of April 2006 is " & hs.GetLastWeekday(dte).ToShortDateString)
```

See Also

IsWeekday
IsWeekend
Weekdays
WeekendDays

See Also

DaylightSavings
DaysInMonth
DaysLeftInMonth
DaysLeftInYear
EvenOddMonth
EvenOddDay
GetSpecialDay
IsSpecialDay
IsWeekday
IsWeekend
Moon
Weekdays
WeekEndDays
WeekNumber
WeekNumberEx
WeeksLeftInYear
WeeksLeftInYearEx

GetSpecialDay

Purpose

This function returns a date object representing the requested special date. e.g. The Third Thursday of November.

Parameters

Parameter: **DOW**
Type: **DayOfWeek (Enum - Integer)**
Description: This is the day of the week value you are looking for. The values for the Enum are as follows:
SUNDAY = 0
MONDAY = 1
TUESDAY = 2
WEDNESDAY = 3
THURSDAY = 4
FRIDAY = 5
SATURDAY = 6
WEEKDAY = 7
WEEKEND_DAY = 8

Parameter: **Instance**
Type: **CD_DAY_IS_Type (Enum - Integer)**
Description: This is the instance day that you wish to retrieve, using these values:
FIRST = 1
SECOND = 2
THIRD = 3
FOURTH = 4
LAST = 5

Parameter: **For Month**
Type: **date**
Description: This date object references the month you are requesting the special date for - the day component of the month is not used. For example, to get the third Thursday in November of 2006, provide a date object set to any day/time in the month of November, 2006.

Optional Parameter: **GetNext**
Type: **Boolean (Default value if not provided = False)**
Description: If the requested special date has already passed and GetNext is True, then the next instance of the requested special day will be returned. (See the example below)

Returns

Return value: **date requested**
Type: **date**
Description: This is a date object with the month, day, year components for the special day requested.

Example

```
Sub Main(parm as object)

  Dim DOW as Integer = 3    ' Wednesday
  Dim Inst as Integer = 3     ' Third instance (e.g. Third Wednesday of the month)
  Dim ForMonth As Date = DateTime.Parse("February 3, 2006")
  Dim dteReturn As Date

  dteReturn = hs.GetSpecialDay(DOW, Inst, ForMonth, False)
  hs.WriteLog("Test","With GetNext False, Result is " & dteReturn.ToShortDateString)

  dteReturn = hs.GetSpecialDay(DOW, Inst, ForMonth, True)
  hs.WriteLog("Test","With GetNext True, Result is " & dteReturn.ToShortDateString)

End Sub
```

The example above returns:

```
 ~!~Test~!~With GetNext False, Result is 2/15/2006
~!~Test~!~With GetNext True, Result is 3/15/2006
```

See Also

IsSpecialDay

See Also

IsSpecialDay

Purpose

This function returns a Boolean (True/False) indicating if a date provided is the special day indicated.

Parameters

Parameter: **In Date**
Type: **date**
Description: This date object references the date you wish to check.

Parameter: **DOW**
Type: **DayOfWeek (Enum - Integer)**
Description: This is the day of the week value you are looking for. The values for the Enum are as follows:
SUNDAY = 0
MONDAY = 1
TUESDAY = 2
WEDNESDAY = 3
THURSDAY = 4
FRIDAY = 5
SATURDAY = 6
WEEKDAY = 7
WEEKEND_DAY = 8

Parameter: **Instance**
Type: **CD_DAY_IS_Type (Enum - Integer)**
Description: This is the instance day that you wish to compare, using these values:
FIRST = 1
SECOND = 2
THIRD = 3
FOURTH = 4
LAST = 5

Parameter: **For Month**
Type: **date**
Description: This date object references the month you are requesting the special date for - the day component of the month is not used. For example, to get the third Thursday in November of 2006, provide a date object set to any day/time in the month of November, 2006.

Returns

Return value: **Is Special**
Type: **Boolean (True/False)**
Description: If In Date matches the special day information indicated with the other three parameters, then Is Special will be True, otherwise False.

Example

```
Sub Main(parm as object)

 Dim DOW as Integer = 3    ' Wednesday
 Dim Inst as Integer = 3     ' Third instance (e.g. Third Wednesday of the month)
 Dim ForMonth As Date = DateTime.Parse("February 3, 2006")
 Dim InDate As Date = DateTime.Parse("February 15, 2006")
```

```
Dim bReturn As Boolean

bReturn = hs.IsSpecialDay(InDate, DOW, Inst, ForMonth)
If bReturn = True Then
   hs.WriteLog("Test","February 15 is the third Wednesday of February, 2006")
Else
   hs.WriteLog("Test","February 15 is the third Wednesday of February, 2006")
End If

End Sub
```

See Also

GetSpecialDay

See Also

DaylightSavings
DaysInMonth
DaysLeftInMonth
DaysLeftInYear
EvenOddMonth
EvenOddDay
GetLastWeekday
GetSpecialDay
IsWeekday
IsWeekend
Moon
Weekdays
WeekEndDays
WeekNumber
WeekNumberEx
WeeksLeftInYear
WeeksLeftInYearEx

IsWeekday

Purpose

This function returns a Boolean (True/False) value indicating whether the date provided is a weekday or not.

Parameters

Parameter: **date**
Type: **date**
Description: This is the date that you wish to check.

Returns

Return value: **Weekday**
Type: **Boolean**
Description: If the date provided falls upon a weekday (Monday through Friday), then this return will be True, otherwise False.

See Also

GetLastWeekday
IsWeekend
Weekdays
WeekendDays

See Also

DaylightSavings
DaysInMonth
DaysLeftInMonth
DaysLeftInYear
EvenOddMonth
EvenOddDay
GetLastWeekday
GetSpecialDay
IsSpecialDay
IsWeekend
Moon
Weekdays
WeekEndDays
WeekNumber
WeekNumberEx
WeeksLeftInYear
WeeksLeftInYearEx

Home > Scripting > Time and Calendar > Calendar Related > IsWeekend

IsWeekend

Purpose

This function returns a Boolean (True/False) value indicating whether the date provided is a weekend day or not.

Parameters

Parameter: **date**
Type: **date**
Description: This is the date that you wish to check.

Returns

Return value: **Weekend**
Type: **Boolean**
Description: If the date provided falls upon a weekend day (Saturday or Sunday), then this return will be True, otherwise False.

See Also

GetLastWeekday
IsWeekday
Weekdays
WeekendDays

See Also

DaylightSavings
DaysInMonth
DaysLeftInMonth
DaysLeftInYear
EvenOddMonth
EvenOddDay
GetLastWeekday
GetSpecialDay
IsSpecialDay
IsWeekday
Moon
Weekdays
WeekEndDays
WeekNumber
WeekNumberEx
WeeksLeftInYear
WeeksLeftInYearEx

Moon

Purpose

This subroutine accepts an input date, and returns into the variables you provide, the values of various moon phase data points including the dates of the new and full moon, the current cycle value, and the moon phase description.

Parameters

Parameter: **dateStart**
Type: **Date**
Description: This is the date for which you wish to retrieve moon phase information.

Parameter: **NewMoon**
Type: **Date**
Description: After the sub-routine completes, this variable will contain the date of the new moon relative to the provided starting date.

Parameter: **FullMoon**
Type: **Date**
Description: After the sub-routine completes, this variable will contain the date of the full moon relative to the provided starting date.

Parameter: **Cycle**
Type: **Integer**
Description: After the sub-routine completes, this variable will contain the value of the moon cycle on the date provided by the starting date.

Parameter: **Description**
Type: **String**
Description: This is the name of the current moon cycle.

Returns

None

Example

```
Sub Main(parm as object)

   Dim dtStart as Date = Now
  Dim NMoon as Date
  Dim FMoon as Date
  Dim CurCycle as Integer
  Dim sDesc as String = ""

     hs.Moon(dtStart, NMoon, FMoon, CurCycle, sDesc)
    hs.WriteLog("Moon","New on " & NMoon.ToShortDateString & ", Full on " & FMoon.ToShortDateString & _
                ", Cycle is " & CurCycle.ToString & " = " & sDesc)

End Sub
```

The above example returns:

```
Moon - New on 8/24/2006, Full on 9/7/2006, Cycle is 24 = Waning Crescent
```

See Also

DaylightSavings
DaysInMonth
DaysLeftInMonth
DaysLeftInYear
EvenOddMonth
EvenOddDay
GetLastWeekday
GetSpecialDay
IsSpecialDay
IsWeekday
IsWeekend
Weekdays
WeekEndDays
WeekNumber
WeekNumberEx
WeeksLeftInYear
WeeksLeftInYearEx

Weekdays

Purpose

This function returns the number of weekdays between two dates, inclusive of the end date.

Parameters

Parameter: **Date Start**
Type: **date**
Description: This is the starting date.

Parameter: **Date End**
Type: **date**
Description: This is the ending date.

Returns

Return value: **Weekdays**
Type: **integer**
Description: This is the number of weekdays between the two dates including the ending date if it is a weekday.

Example

```
Sub Main(parm as object)

   Dim dtStart as Date = DateTime.Parse("8/1/2006")
  Dim dtEnd as Date = DateTime.Parse("8/10/2006")
  Dim iResult as Integer

  iResult = hs.Weekdays(dtStart, dtEnd)
  hs.WriteLog("Weekdays","There are " & iResult.ToString & " weekdays between the dates.")

End Sub
```

The above example returns this result:

Weekdays - There are 7 weekdays between the dates.

See Also

GetLastWeekday
IsWeekday
IsWeekend
WeekendDays

See Also

DaylightSavings
DaysInMonth
DaysLeftInMonth
DaysLeftInYear
EvenOddMonth
EvenOddDay
GetLastWeekday
GetSpecialDay
IsSpecialDay
IsWeekday
IsWeekend
Moon
WeekEndDays
WeekNumber
WeekNumberEx
WeeksLeftInYear
WeeksLeftInYearEx

WeekEndDays

Purpose

This function returns the number of weekend days between two dates, inclusive of the end date.

Parameters

Parameter: **Date Start**
Type: **date**
Description: This is the starting date.

Parameter: **Date End**
Type: **date**
Description: This is the ending date.

Returns

Return value: **Weekdays**
Type: **integer**
Description: This is the number of weekend days between the two dates including the ending date if it is a Saturday or Sunday.

Example

```
Sub Main(parm as object)

  Dim dtStart as Date = DateTime.Parse("8/1/2006")
 Dim dtEnd as Date = DateTime.Parse("8/13/2006")
 Dim iResult as Integer

 iResult = hs.WeekEndDays(dtStart, dtEnd)
 hs.WriteLog("WeekEndDays","There are " & iResult.ToString & " weekend days between the dates.")

End Sub
```

The above example returns this result:

WeekEndDays - There are 4 weekend days between the dates.

See Also

GetLastWeekday
IsWeekday
IsWeekend
Weekdays

See Also

DaylightSavings
DaysInMonth
DaysLeftInMonth
DaysLeftInYear
EvenOddMonth
EvenOddDay
GetLastWeekday
GetSpecialDay
IsSpecialDay
IsWeekday
IsWeekend
Moon
Weekdays
WeekNumber
WeekNumberEx
WeeksLeftInYear
WeeksLeftInYearEx

Home > Scripting > Time and Calendar > Calendar Related > WeekNumber

WeekNumber

Purpose

This function returns the week number of the year for the given date, and assumes the first full week starting on Sunday of the year as Week 1. For other options on the first week of the year, use WeekNumberEx.

Parameters

Parameter: **In Date**
Type: **date**
Description: This is the date for which you wish to know the week number.

Returns

Return value: **WeekNumber**
Type: **short integer**
Description: This is the week number of the year for the given date.

See Also

DaysInMonth
DaysLeftInMonth
DaysLeftInYear
WeeksLeftInYear
WeeksLeftInYearEx
WeekNumberEx

See Also

DaylightSavings
DaysInMonth
DaysLeftInMonth
DaysLeftInYear
EvenOddMonth
EvenOddDay
GetLastWeekday
GetSpecialDay
IsSpecialDay
IsWeekday
IsWeekend
Moon
Weekdays
WeekEndDays
WeekNumberEx
WeeksLeftInYear
WeeksLeftInYearEx

WeekNumberEx

Purpose

This function returns the week number of the year for the given date, just like WeekNumber, except that you can specify the conditions for determining the first week of the year.

Parameters

Parameter: **In Date**
Type: **date**
Description: This is the date for which you wish to know the week number.

Parameter: **Week Mode**
Type: **Integer**
Description: This specifies how the first week of the year is determined, according to the following table:

Week Mode Value	Result

1	The first week of the year starts with the first calendar day of the year, regardless of the day of the week it falls upon.
4	The first week of the year is determined by the first week with at least four days in the new year.
(Anything Else)	The first week of the year is determined by the first full week starting on Sunday in the new year.

Returns

Return value: **WeekNumber**
Type: **Integer**
Description: This is the week number of the year for the given date and Week Mode.

See Also

DaysInMonth
DaysLeftInMonth
DaysLeftInYear
WeeksLeftInYear
WeeksLeftInYearEx
WeekNumber

See Also

DaylightSavings
DaysInMonth
DaysLeftInMonth
DaysLeftInYear
EvenOddMonth
EvenOddDay
GetLastWeekday
GetSpecialDay
IsSpecialDay
IsWeekday
IsWeekend
Moon
Weekdays
WeekEndDays
WeekNumber
WeeksLeftInYear
WeeksLeftInYearEx

WeeksLeftInYear

Purpose

This function returns the number of weeks left in the current year based upon the first week of the year being the first full week starting on a Sunday. For other starting week options, see WeeksLeftInYearEx.

Parameters

None.

Returns

Return value: **Weeks Left**
Type: **Integer**
Description: This number represents the number of weeks remaining in the current year based upon the first week being the first full week starting on Sunday of the year.

See Also

DaysInMonth
DaysLeftInMonth
DaysLeftInYear
WeekNumber
WeekNumberEx
WeeksLeftInYearEx

See Also

DaylightSavings
DaysInMonth
DaysLeftInMonth
DaysLeftInYear
EvenOddMonth
EvenOddDay
GetLastWeekday
GetSpecialDay
IsSpecialDay
IsWeekday
IsWeekend
Moon
Weekdays
WeekEndDays
WeekNumber
WeekNumberEx
WeeksLeftInYearEx

WeeksLeftInYearEx

Purpose

This function returns the number of weeks left in the current year, based upon the starting week mode provided as a parameter.

Parameters

Parameter: **Week Mode**
Type: **integer (Optional)**
Description: This specifies how the first week of the year is determined, according to the following table:

Week Mode Value	Result
1	The first week of the year starts with the first calendar day of the year, regardless of the day of the week it falls upon.
4	The first week of the year is determined by the first week with at least four days in the new year.
(Anything Else)	The first week of the year is determined by the first full week starting on Sunday in the new year.

Returns

Return value: **Weeks Left**
Type: **short**
Description: This number represents the number of weeks remaining in the current year as determined by the week mode parameter.

See Also

DaysInMonth
DaysLeftInMonth
DaysLeftInYear
WeekNumber
WeekNumberEx

WeeksLeftInYear

See Also

DaylightSavings
DaysInMonth
DaysLeftInMonth
DaysLeftInYear
EvenOddMonth
EvenOddDay
GetLastWeekday
GetSpecialDay
IsSpecialDay
IsWeekday
IsWeekend
Moon
Weekdays
WeekEndDays
WeekNumber
WeekNumberEx
WeeksLeftInYear

Text-To-Speech and Media

Text to Speech (TTS) and media are handled independently by the speaker clients. It is possible to have a media file playing and at the same time, have TTS being generated. If the computer that the speaker client is installed on only has one sound output, then both sounds are heard at the same time, mixed together. If the system has more than one sound output/resource, and Windows Media Player is set to use a different one than the default for audio, then it is possible to have the output from TTS and Media functions go their separate ways.

The TTS channel can also play WAV media files through PlayWavFile or PlayWavFileVol, so it is possible to have WAV audio play on the TTS channel in the event that the TTS channel and MEDIA channel are routed out separate sound devices.

This section covers the script commands for generating TTS, playing/controling media files, and controlling speaker clients.

HSTouch clients, as a speaker client, are more limited in their scope - they can only process TTS and will ignore the media related commands in this section.

In This Section

GetInstanceList
IsSpeakerBusy
SpeakToFile
Speaker Client Global Audio
Media Only Procedures
Text-to-Speech Only Procedures

See Also

About Scripts
Applications and Plugins
Computer
Devices
Email
Events
Internet
Phone
Scripts
Speech Recognition
Strings, Global Variables, and Encryption
Time and Calendar

GetInstanceList

Purpose

This function retrieves a comma separated list of host:instance names for Speaker client instances currently connected to HomeSeer.

Parameters

None.

Returns

Return value: **instance list**
Type: **string**
Description: The returned instance list is a comma separated list of host:instance pairs as in this example:

Bandit:Default,Johnny:Default,Race:Music,Race:Default

See Also

IsSpeakerBusy
SpeakToFile
Speaker Client Global Audio
Media Only Procedures
Text-to-Speech Only Procedures

IsSpeakerBusy

Purpose

This function can let you know if a specific speaker client (host or host:instance) is currently busy speaking or playing WAV audio.

Parameters

Parameter: **Host (Optional)**
Type: **String**
Description: Leaving this a null string will return the busy status for the first instance HomeSeer finds, otherwise use the hostname of the computer you are interested in determining the busy status of. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

Return value: **busy status**
Type: **boolean**
Description: TRUE indicates that the speaker application instance is busy.

See Also

GetInstanceList
SpeakToFile
Speaker Client Global Audio
Media Only Procedures
Text-to-Speech Only Procedures

SpeakToFile

Purpose

This function speaks some text and saves the result in a WAV file.

Parameters

Parameter: **Text**
Type: **String**
Description: This is the string you want to speak.

Parameter: **Voice**
Type: **String**
Description: This is the name of the voice you want to use for speaking. This string must match the voice name exactly. Voice names can be found in the Speaker Client. If the name is omitted, the default voice as specified in the computer's speech control panel is used.

Parameter: **Filename**
Type: **String**
Description: This is the full path to the file where the voice output will be saved.

Returns

None.

Example

```
sub main()

    hs.SpeakToFile "Hello from a file!", "ATT DTNV 1.3 Crystal","c:\voice.wav"

end sub
```

see Also

Using Replacement Variables

See Also

GetInstanceList
IsSpeakerBusy
Speaker Client Global Audio
Media Only Procedures
Text-to-Speech Only Procedures

Speaker Client Global Audio

In This Section

SetVolume
GetVolume
GetMuteStatus
GetPauseStatus
MuteAudio
PauseAudio
UnMuteAudio
UnPauseAudio

See Also

GetInstanceList
IsSpeakerBusy
SpeakToFile
Media Only Procedures
Text-to-Speech Only Procedures

SetVolume

Purpose

This function sets the master volume of the system sound device that the speaker client(s) are using. This can be used to set the volume of the text-to-speech output. The volume level must be in a range between 0 and 100, where 100 is the maximum volume.

To change the volume for the MEDIA functions, use MediaVolume.

Parameters

Parameter: **Level**
Type: **Integer**
Description: This is the volume level, from 0 to 100.

Parameter: **Host (Optional)**
Type: **String**
Description: Leaving this a null string will apply the command to the first instance HomeSeer finds, otherwise use the hostname of the computer for this command. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

None.

Example

```
sub main()

   hs.SetVolume 90
   hs.speak "I am speaking louder",TRUE
   hs.SetVoume 20, "Kitchen"
   hs.speak "I am speaking softer on the Kitchen computer than on the others.",TRUE

end sub
```

See Also

GetVolume
GetMuteStatus
GetPauseStatus
MuteAudio
PauseAudio
UnMuteAudio
UnPauseAudio

GetVolume

Purpose

This function returns the volume level of an instance of the Speaker client program running on a computer.

Parameters

Parameter: **Host (Optional)**
Type: **String**
Description: Leaving this a null string will return the volume level for the first instance HomeSeer finds, otherwise use the hostname of the computer you are interested in determining the volume level of. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

Return value: **Volume level**
Type: **Integer**
Description: The volume level is returned using a 0-100 scale, 100 being full volume.

See Also

SetVolume
GetMuteStatus
GetPauseStatus
MuteAudio
PauseAudio
UnMuteAudio
UnPauseAudio

GetMuteStatus

Purpose

This function returns the mute status of a specific speaker client (host or host:instance).

Parameters

Parameter: **Host (Optional)**
Type: **String**
Description: Leaving this a null string will return the status for the first instance HomeSeer finds, otherwise use the hostname of the computer you are interested in determining the listening status of. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

Return value: **Mute Status**
Type: **Boolean**
Description: TRUE indicates that the speaker app instance is muted.

See Also

SetVolume
GetVolume
GetPauseStatus
MuteAudio
PauseAudio
UnMuteAudio
UnPauseAudio

GetPauseStatus

Purpose

This function returns the "pause" status of a specific speaker client (host or host:instance).

Parameters

Parameter: **Host (Optional)**
Type: **String**
Description: Leaving this a null string will return the status for the first instance HomeSeer finds, otherwise use the hostname of the computer you are interested in determining the pause status of. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

Return value: **Pause Status**
Type: **Integer**
Description: Bit encoded value indicating the status as follows:

Bit Values	Status
Bit 1 = 0	No Wavefile Instance
Bit 1 = 1	Wavefile Present
Bit 2 = 0	Wavefile Paused
Bit 2 = 1	Wavefile Playing
Bit 3 = 0	TTS Present
Bit 3 = 1	No TTS Present
Bit 4 = 0	TTS is currently speaking
Bit 4 = 1	TTS is not speaking

See Also

SetVolume
GetVolume
GetMuteStatus
MuteAudio
PauseAudio
UnMuteAudio
UnPauseAudio

MuteAudio

Purpose

This function mutes all speech and audio of a specific speaker client (host or host:instance).

Parameters

Parameter: **Host (Optional)**
Type: **String**
Description: Leaving this a null string will mute the first instance HomeSeer finds, otherwise use the hostname of the computer you are interested in muting. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

None.

See Also

SetVolume
GetVolume
GetMuteStatus
GetPauseStatus
PauseAudio
UnMuteAudio
UnPauseAudio

PauseAudio

Purpose

This function pauses the audio currently playing at a specific speaker client (host or host:instance).

Parameters

Parameter: **Host (Optional)**
Type: **String**
Description: Leaving this a null string will pause the audio for the first instance HomeSeer finds, otherwise use the hostname of the computer you are interested in pausing audio on. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

None.

See Also

SetVolume
GetVolume
GetMuteStatus
GetPauseStatus
MuteAudio
UnMuteAudio
UnPauseAudio

UnMuteAudio

Purpose

This function resumes all speech and audio of a specific speaker client (host or host:instance).

Parameters

Parameter: **Host (Optional)**
Type: **String**
Description: Leaving this a null string will mute the first instance HomeSeer finds, otherwise use the hostname of the computer you are interested in muting. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

None.

See Also

SetVolume
GetVolume
GetMuteStatus
GetPauseStatus
MuteAudio
PauseAudio
UnPauseAudio

UnPauseAudio

Purpose

This function resumes the audio currently playing at a specific speaker client (host or host:instance).

Parameters

Parameter: **Host (Optional)**
Type: **String**
Description: Leaving this a null string will pause the audio for the first instance HomeSeer finds, otherwise use the hostname of the computer you are interested in pausing audio on. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

None.

See Also

SetVolume
GetVolume
GetMuteStatus
GetPauseStatus
MuteAudio
PauseAudio
UnMuteAudio

Media Only Procedures

In This Section

MediaFilename
MediaPlay
MediaPause
MediaMute
MediaIsPlaying
MediaStop
MediaUnPause
MediaVolume

See Also

GetInstanceList
IsSpeakerBusy
SpeakToFile
Speaker Client Global Audio
Text-to-Speech Only Procedures

MediaFilename

Purpose

This is a read/write property. This function sets the file name that is to played using the speaker client. Call MEDIAPlay to actually start playing the selection.

This property may be read to get the selection currently playing.

Parameters

Parameter: **filename**
Type: **string**
Description: This sets the file name of the media selection to play. The file name may be any valid file supported by the Windows® Media Player.

Parameter: **host** (optional)
Type: **string**
Description: Leaving this a null string will apply the command to the first instance HomeSeer finds, otherwise use the hostname of the computer for this command. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

None.

See Also

MediaPlay
MediaPause
MediaMute
MediaIsPlaying
MediaStop
MediaUnPause
MediaVolume

MediaPlay

Purpose

This function starts playing the selection as specified with the hs.MEDIAFilename property.

Parameters

Parameter: **filename** (optional)
Type: **string**
Description: This is the path and filename of the file to be played. If it is omitted here, it must have been previously set using the MEDIAFilename property.

Parameter: **host** (optional)
Type: **string**
Description: Leaving this a null string will apply the command to the first instance HomeSeer finds, otherwise use the hostname of the computer for this command. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

None.

See Also

MediaFilename
MediaPause
MediaMute
MediaIsPlaying
MediaStop
MediaUnPause
MediaVolume

MediaPause

Purpose

This function instructs the Windows® Media Player to pause the currently playing selection. The selection may be resumed by calling the hs.MediaPlay function.

Parameters

Parameter: **host** (optional)
Type: **string**
Description: Leaving this a null string will apply the command to the first instance HomeSeer finds, otherwise use the hostname of the computer for this command. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

None.

See Also

MediaFilename
MediaPlay
MediaMute
MediaIsPlaying
MediaStop
MediaUnPause
MediaVolume

MediaMute

Purpose

This function mutes the media selection that's currently playing. The selection continues to play, but sound is not heard.

Parameters

Parameter: **mute**
Type: **boolean**
Description: Use TRUE to mute the selection and FALSE to unmute it.

Parameter: **host** (optional)
Type: **string**
Description: Leaving this a null string will apply the command to the first instance HomeSeer finds, otherwise use the hostname of the computer for this command. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

None.

Example

```
sub main()

    ' mute the Windows Media Player
    hs.MediaMute TRUE

end sub
```

See Also

MediaFilename
MediaPlay
MediaPause
MediaIsPlaying
MediaStop
MediaUnPause
MediaVolume

MediaIsPlaying

Purpose

This function checks if the media player is currently playing a selection.

Parameters

Parameter: **host** (optional)
Type: **string**
Description: Leaving this a null string will apply the command to the first instance HomeSeer finds, otherwise use the hostname of the computer for this command. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

Return value: **status**
Type: **boolean**
Description: This returns TRUE if a media selection is currently playing and the sound card is most likely busy, and returns FALSE if a media selection is not playing and the sound is most likely free.

See Also

MediaFilename
MediaPlay
MediaPause
MediaMute
MediaStop
MediaUnPause
MediaVolume

MediaStop

Purpose

This function instructs the Windows® Media Player to stop playing the current selection.

Parameters

Parameter: **host** (optional)
Type: **string**
Description: Leaving this a null string will apply the command to the first instance HomeSeer finds, otherwise use the hostname of the computer for this command. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

None.

See Also

MediaFilename
MediaPlay
MediaPause
MediaMute
MediaIsPlaying
MediaUnPause
MediaVolume

MediaUnPause

Purpose

This function instructs the Windows® Media Player to resume the currently playing selection.

Parameters

Parameter: **host** (optional)
Type: **string**
Description: Leaving this a null string will apply the command to the first instance HomeSeer finds, otherwise use the hostname of the computer for this command. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

None.

See Also

MediaFilename
MediaPlay
MediaPause
MediaMute
MediaIsPlaying
MediaStop
MediaVolume

MediaVolume

Purpose

This is a read/write property. It sets and gets the current volume level of the playing media selection.

Parameters

Parameter: **Level**
Type: **Integer** (property)
Description: This sets the volume level. 100=full volume and 0 is the lowest volume.

Parameter: **Host** (optional)
Type: **String**
Description: Leaving this a null string will apply the command to the first instance HomeSeer finds, otherwise use the hostname of the computer for this command. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

None.

Example

```
sub main()

    ' get the current volume level
    dim level
    level = hs.MediaVolume

    ' set the volume to full
    hs.MediaVolume = 100

end sub
```

See Also

MediaFilename
MediaPlay
MediaPause
MediaMute
MediaIsPlaying
MediaStop
MediaUnPause

Text-to-Speech Only Procedures

In This Section

Speak
SpeakEx
SpeakProxy
GetVoiceName
MuteSpeech
SetSpeakingSpeed
SetVoice
StopSpeaking
PlayWavFile
PlayWavFileVol

See Also

GetInstanceList
IsSpeakerBusy
SpeakToFile
Speaker Client Global Audio
Media Only Procedures

Speak

Purpose

This function speaks some text.

Parameters

Parameter: **Text**
Type: **String**
Description: This is the string you want to speak. It may also be the complete path to a WAV file to be played.

Parameter: **Wait (Optional)**
Type: **Boolean**
Description: If set to TRUE, the function will not return until the system finishes speaking. This is useful if you are switching between speaking and listening. You cannot listen and speak at the same time on some systems. If this parameter is missing, the system will not wait.

Parameter: **Host (Optional)**
Type: **String**
Description: Leaving this a null string will apply the command to the first instance HomeSeer finds, otherwise use the hostname of the computer for this command. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

None.

Example

```
Sub Main()

    ' speak and wait
    hs.speak "hello there", True
    hs.speak "Hello people in the kitchen.", True, "Kitchen:*"

End Sub
```

See Also

Using Replacement Variables

See Also

SpeakEx
SpeakProxy
GetVoiceName
MuteSpeech
SetSpeakingSpeed
SetVoice
StopSpeaking
PlayWavFile
PlayWavFileVol

SpeakEx

Purpose

This function speaks some text and sends the output to the indicated output device. This function can be used to speak out other sound devices other than the normal sound card. For systems with multiple sound cards, this function can be used to select the specific card.

Parameters

Parameter: **device**
Type: **integer**
Description: This is the device number of the output device. Device 0 is usually the computer speakers and the default sound card.

Parameter: **text**
Type: **string**
Description: This is the text you want to speak.

Parameter: **wait**
Type: **boolean**
Description: If set to TRUE, the function will not return until the system finishes speaking. This is useful if you are switching between speaking and listening. You cannot listen and speak at the same time on some systems. If this parameter is missing, the system will not wait.

Parameter: **Host (Optional)**
Type: **String**
Description: Leaving this a null string will apply the command to the first instance HomeSeer finds, otherwise use the hostname of the computer for this command. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

None.

See Also

Using Replacement Variables

See Also

Speak
SpeakProxy
GetVoiceName
MuteSpeech
SetSpeakingSpeed
SetVoice
StopSpeaking
PlayWavFile
PlayWavFileVol

SpeakProxy

Purpose

This function speaks some text after being handled by a speaker proxy program or plug-in.

- This command passes along speak commands received from HomeSeer as a registered speaker proxy handler, and this is where the values for the parameters are provided. Therefore, this command is generally NOT used by a script and is primarily for plug-ins and applications.

Parameters

Parameter: **Device**
Type: **Integer**
Description: This is the sound device ID number for the TTS to be spoken at.

Parameter: **Text**
Type: **String**

Description: This is the string you want to speak. It may also be the complete path to a WAV file to be played.

Parameter: **Wait**
Type: **Boolean**
Description: If set to TRUE, the function will not return until the system finishes speaking. This is useful if you are switching between speaking and listening. You cannot listen and speak at the same time on some systems. If this parameter is missing, the system will not wait.

Parameter: **Host (Optional)**
Type: **String**
Description: Leaving this a null string will apply the command to the first instance HomeSeer finds, otherwise use the hostname of the computer for this command. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

None.

See Also

Speak
SpeakEx
GetVoiceName
MuteSpeech
SetSpeakingSpeed
SetVoice
StopSpeaking
PlayWavFile
PlayWavFileVol

GetVoiceName

Purpose

This function returns the voice name of a specific speaker client (host or host:instance).

Parameters

Parameter: **Host (Optional)**
Type: **String**
Description: Leaving this a null string will return the voice name for the first instance HomeSeer finds, otherwise use the hostname of the computer you are interested in determining the voice name being used. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

Return value: **voice name**
Type: **string**

See Also

Speak
SpeakEx
SpeakProxy
MuteSpeech
SetSpeakingSpeed
SetVoice
StopSpeaking
PlayWavFile
PlayWavFileVol

MuteSpeech

Purpose

This function temporarily mutes the speech output. By setting this property to FALSE, all speech output is silenced until this property is set back to TRUE. This mutes ALL speech, including speech generated from scripts.

This is a read/write property.

Parameters

Parameter: **Mode**
Type: **Boolean**
Description: Use TRUE to have speech output silenced and FALSE to have it enabled.

Returns

None.

Example

```
' stop all speech output
sub main()
   hs.MuteSpeech = TRUE
end sub

' enable all speech output
sub main()
   hs.MuteSpeech = FALSE
end sub
```

See Also

Speak
SpeakEx
SpeakProxy
GetVoiceName
SetSpeakingSpeed
SetVoice
StopSpeaking
PlayWavFile
PlayWavFileVol

SetSpeakingSpeed

Purpose

This function sets the rate of HomeSeer's speech.

Parameters

Parameter: **Speed**
Type: **Integer**
Description: This is the speed parameter to be set. The range is -10 to 10. A value of zero is the normal rate of speaking.

Parameter: **Host (Optional)**
Type: **String**
Description: Leaving this a null string will set the speaking speed for the first instance HomeSeer finds, otherwise use the hostname of the computer you are interested in setting the speaking speed on. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

Return value: **previous speed** or **99**
Type: **integer**
Description: This returns the previous speed setting or returns 99 if the input parameter was invalid. This is useful for returning the speaking speed to

its previous value after making an adjustment.

Example

```
Sub Main()

    Dim iOldSpeed
    Dim iNothing

    hs.Speak "This is the rate at which I am currently speaking."

    iOldSpeed = hs.SetSpeakingSpeed(8)

    hs.Speak "Now I am talking very fast like I just drank two pots of coffee."

    iNothing = hs.SetSpeakingSpeed(iOldSpeed)

End Sub
```

See Also

Speak
SpeakEx
SpeakProxy
GetVoiceName
MuteSpeech
SetVoice
StopSpeaking
PlayWavFile
PlayWavFileVol

SetVoice

Purpose

This command changes the voice of a speaker client instance to the voice name provided.

Parameters

Parameter: **VoiceName**
Type: **String**
Description: This is the voice name string of the voice you wish to change the speaker client to use - it is not case sensitive but must match one of the voice names in your system. (See the Speaker Client for a list of voice names.)

Parameter: **Host (Optional)**
Type: **String**
Description: Leaving this a null string will change the voice for the first instance HomeSeer finds, otherwise use the hostname of the computer you are interested in changing the voice of. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

Return value: **return status**
Type: **integer (.NET Short)**
Description: Zero (0) means the voice was not found, One (1) indicates success.

See Also

Speak
SpeakEx
SpeakProxy
GetVoiceName
MuteSpeech
SetSpeakingSpeed
StopSpeaking
PlayWavFile
PlayWavFileVol

StopSpeaking

Purpose

This function causes any speaking to stop immediately.

Parameters

Parameter: **Host (Optional)**
Type: **String**
Description: Leaving this a null string will apply the command to the first instance HomeSeer finds, otherwise use the hostname of the computer for this command. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Returns

None.

Example

```
hs.StopSpeaking
```

See Also

Speak
SpeakEx
SpeakProxy
GetVoiceName
MuteSpeech
SetSpeakingSpeed
SetVoice
PlayWavFile
PlayWavFileVol

PlayWavFile

Purpose

This function plays a specific WAV file out the default audio device. For more control over playing WAV files, see PlayWavFileEx.

Parameters

Parameter: **FileName**
Type: **String**
Description: This is the complete path to the WAV file to play.

Parameter: **Host (Optional)**
Type: **String**
Description: Leaving this a null string will apply the command to the first instance HomeSeer finds, otherwise use the hostname of the computer for this command. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Parameter: **Wait (Optional)**
Type: **Boolean**
Description: Setting this to True will cause the command to wait until the WAV file is done playing before continuing. By default, it will not wait.

Returns

None.

See Also

Speak
SpeakEx
SpeakProxy
GetVoiceName
MuteSpeech
SetSpeakingSpeed
SetVoice
StopSpeaking
PlayWavFileVol

PlayWavFileVol

Purpose

This function plays a WAV file and allows playing the WAV file in the background and setting the volume level.

Parameters

Parameter: **Filename**
Type: **String**
Description: This is the complete path to the WAV file to play.

Parameter: **volume (left)**
Type: **Integer**
Description: This is the volume level to use when playing. The range is 0 to 100. Set the value to -1 if you want to use the currently set volume level. In previous versions of HomeSeer this was the LEFT volume level only - note that this is now the one and only volume level.

Parameter: **volume (right)**
Type: **Integer**
Description: This parameter is obsolete and remains for backward compatibility with previous versions of HomeSeer.

Parameter: **Host** (optional)
Type: **String**
Description: Leaving this a null string will apply the command to the first instance HomeSeer finds, otherwise use the hostname of the computer for this command. If more than one instance of the Speaker application is running on "host" then you may need to specify the instance as well in the format host:instance.

Parameter: **Wait**
Type: **Boolean**
Description: Use TRUE to not return until the WAV file has finished playing and FALSE to play the WAV file in the background. The function returns immediately.

Returns

None.

See Also

Speak
SpeakEx
SpeakProxy
GetVoiceName
MuteSpeech
SetSpeakingSpeed
SetVoice
StopSpeaking
PlayWavFile

Index