STSTray 2.0

The scriptable tray icon utility

User's Guide

Developed by

1511 Fairway Street, Eau Claire, WI 54701
Phone: (715) 831-8988 Fax: (775) 429-1203
Web: http://www.sonsothunder.com/
Email: ststray@sonsothunder.com

STSTray, the Scriptable Tray Icon Utility

Table of Contents

STSTray 2.0

Introduction

STSTray is a solution that can be used with any application[1] so long as it can read and write plain text files. At its simplest, it is an application that allows you to display an icon in the Windows system tray. But that's not all that it is able to do, it can also:

- Display a menu when the user clicks on the icon, and take actions when the user selects an item from the menu, including communicating back to your application what the user selected
- Display a tooltip when the mouse passes over the icon
- Display a balloon under XP on demand
- Flash the icon at a user-settable time interval
- Switch back and forth between two icons at a user-settable time interval
- Display an "answer" dialog box with different settings, and capture the selection made by the user to pass back to your application
- Write a file to disk
- Launch a program
- Delete a file from disk *(new in 2.0)*
- Prepare an email in your local email program *(new in 2.0)*
- Open a web browser and surf to a specific web site *(new in 2.0)*
- Download files from the Internet *(new in 2.0)*
- Check a web server and notify users when updates to your product is available *(new in 2.0)*
- Perform actions based on a timed interval, either once, or every X period of time *(new in 2.0)*
- Perform actions on trapped events in one of two ways: once only, or every time the event occurs *(new in 2.0)*
- and more…

How would you use this? Here are a few scenarios where this utility can come in handy:

- **To Let Everyone Know When a New Product Update is Available:** You have distributed STSTray to your customers along with your product. When an update to your product is available, you simply update a single file on your web server, and then everyone gets notified via STSTray (perhaps a balloon or alert box, which when clicked would go to the web and see your info on the updated product).
- **For Notification:** You have a program that acts like a background process and needs to notify the user when the job is done. It could flash the icon in the tray, and when the user clicked on it, it could display a custom message.

[1] STSTray was originally designed with support for xTalk programming environments such as Revolution, MetaCard, etc. in mind, but its application is quite universal.

- **As a Reminder:** Your program creates scheduled events, and you want to remind your user when something is scheduled to occur. It could show a balloon or flash the icon in the tray, and when the user clicked on it, it could display a custom message.
- **As a Launcher:** You want to give your users a nice way to launch your program, or to set certain preferences. You could install an icon in their tray, and they could click and display a menu allowing them to launch your program or display a message.

Technical Support/Contact Info

STSTray was developed by Sons of Thunder Software (http://www.sonsothunder.com/). If there are any technical support issues, comments or questions, please send them via email to ststray@sonsothunder.com.

System Requirements

This utility was written in Visual Basic 6.0, and will run on Windows 95 and higher, and requires version 5.0 of the VB library or higher on the target system. This file is called either `msvbvm50.dll` or `msvbvm60.dll` and exists in the "System" directory for the target Windows platform (`c:\windows\system`, `c:\windows\system32` or `c:\winnt\system32`). This library is very common, and will most likely already be present on the target system. It is recommended that your installer check first to see if it exists before installing it. Included in the STSTray package is version 6.0 of the library (`msvbvm60.dll`) that you can install as needed.

Installing STSTray

STSTray only requires that the VB Library be present on the machine, and so you can "install" STSTray anywhere you like, so long as it is on a writeable volume (STSTray uses special files to control its behavior). If the VB Library is not installed, you can install it either in the same directory as STSTray, or you can install it into the Windows, WinNT or System32 directory. Once installed, all you need to is launch STSTray with the proper script file.

Scripts

Scripts are files that tell STSTray what to do, how to look, and how to act. The scripting language is called *TrayScript* and is very similar to other xTalk languages in products such as HyperCard, SuperCard, ToolBook, MetaCard, etc.). The TrayScript language has only a handful of commands and events. Luckily, this handful will accommodate most people's needs. TrayScript is divided into two main pieces – commands that cause actions to occur, and event handlers ("handlers") that are triggered when an event occurs, and in turn execute one or more commands.

For example, when a TrayScript file is opened by STSTray to be processed, it looks for the `on openScript` handler in that file and processes the commands within that handler. This simple script would bring up an alert dialog box that says "Hello":

```
on openScript
  answer "Hello"
end openScript
```

Additional handlers in the TrayScript file determine what should occur when a certain event happens. For example, the following TrayScript not only displays an alert dialog that says "Hello" when the script is run, but also sets up an event that traps the user double-clicking on the icon in the tray; when the user does this, it launches the Notepad application:

```
on openScript
  answer "Hello"
end openScript

on doubleClick
  launch "c:\windows\notepad.exe"
end doubleClick
```

Of course more detailed explanations of each command and event are located in the TrayScript Language Guide section of this document.

There are two different kinds of TrayScript files: *boot scripts* (which are executed when STSTray launches (see Launching STSTray, below)) and *drop scripts* (which are executed after STSTray has been launched (see Polling Mode, below)).

The "Active Handlers"

Each TrayScript file that is opened will execute what is in the `on openScript` handler within the file, but all other handlers are loaded into memory, and are triggered later on when a matching event occurs. These handlers that remain in memory are called the *Active Handlers*.

STSTray allows for *multiple* TrayScript files to be opened during its operation (through the use of *drop scripts* (see Polling Mode, below)), and if the handlers that are read in at that time are *different* than those currently loaded in the Active Handlers, they will be *added* to the Active Handlers. If one or more handlers are read in that are the *same* as any handlers currently in the Active Handlers, the new handlers *replace* the old one(s) in the Active Handlers.

For example, suppose you launched STSTray and opened one TrayScript file that contained the following:

```
on doubleClick
  launch "c:\windows\notepad.exe"
end doubleClick
```

The Active Handlers would only contain the code above (since nothing had been previously loaded). If you then opened another TrayScript file that contained:

```
on leftClick
  answer "Hello there!"
end leftClick
```

The Active Handlers would contain *both* handlers:

```
on doubleClick
  launch "c:\windows\notepad.exe"
end doubleClick

on leftClick
  answer "Hello there!"
end leftClick
```

If you later opened a TrayScript file that contained:

```
on leftClick
  beep
end leftClick
```

The old `leftClick` handler would be replaced by the new one, and the Active Handlers would look like this:

```
on doubleClick
  launch "c:\windows\notepad.exe"
end doubleClick
```

```
on leftClick
  beep
end leftClick
```

This can be very useful for replacing outdated scripts, or to cause a popup menu that is displayed from the icon on the system tray to be changed with a different menu.

Note: If you ever want to remove a handler from the Active Handlers, you actually need to load up an empty handler (one that has no commands) through the use of a drop script. So using the example above, if you wanted to have nothing happen when the user clicked the tray icon with the left mouse button (`leftClick`), you could load a TrayScript file that looked like this:

```
on leftClick
end leftClick
```

Commenting Scripts/White Space

You can comment out a script line by using a double-hyphen (`--`) or a pound sign (#) before a line of script. Commented script lines will be ignored by the interpreter. Blank lines and any spaces or tabs that preceded a line will be ignored as well, so please take advantage of this for readability in your scripts.

Launching STSTray

The STSTray is a small (112K) executable (`ststray.exe`) that you would launch from your application,[2] or during the startup process of your computer (by placing a shortcut in the Startup folder in the Programs folder in the Start menu).

When STSTray launches, it will look for a TrayScript file called `boot.scp` (called the "boot script"). If it finds one, it will open up and examine its contents, and execute everything found in the `on openScript` handler and store the rest of the script in the Active Handlers (so it can take actions later based on matching events).

Launch time is a good time to set up the menu (if you want one (see the `insert menu` command)), change the icon to an icon of your choosing (if you want something other than the STSTray icon (see the `set icon` command)), and/or set the tooltip for the icon (see the `set tooltip` command).

After it has read the script into memory and executed it, it will go into *polling mode*.

Polling Mode

After STSTray has launched and processed the boot script file, it will go into polling mode, checking every 500 milliseconds[3] to see if there are any files with a `.scp` extension that are in the same directory as the executable (also called a "drop script"). If there are, it will do the following for each drop script that it finds:

1. Open the `.scp` file and read the contents into memory.
2. Execute any commands found in the `on openScript` handler (if there is one).
3. Store the rest of the handlers found in the Active Handlers, either adding them or overwriting handlers that are already loaded.
4. Delete the `.scp` file and go back to polling.

This can be used to communicate with STSTray from your application.

[2] For MetaCard/Revolution users, this would be either the `launch`, `shell` or `open process` commands.
[3] This interval can be changed by using the `set polling` command.

Example for MetaCard/Revolution Users

For example, supposed you wanted to cause a notification balloon to be displayed (under Windows XP or greater). you could execute this code (assuming STSTray is installed in the default location):

```
on showTrayBalloon
  put "c:\Program Files\STS\STSTray\showballoon.scp" into tScriptFile
  put "on openScript" & cr & "show info balloon with" && \
    q("Ready to go?") && "titled" && q("Question:") & cr & \
    "end openScript" into url("file:" & tScriptFile)
end showTrayBalloon

function q what
  return quote & what & quote
end q
```

Example for Visual Basic Users

To do this same thing in Visual Basic, you could use the following code:

```
Private Sub ShowTrayBalloon()
  strScriptFile = "C:\Program Files\STS\STSTray\showballoon.scp"
  strScript = "on openScript" & vbCrLf & "show info balloon with " & _
    q("Ready to go?") & " titled " & q("Question:") & vbCrLf & _
    "end openScript"
  Open strScriptFile for Output as #1
  Print #1,strScript
  Close #1
End Sub

Private Function q(ByVal strData as String)
  q = Chr(34) & strData & Chr(34)
End Function
```

The "Output File"

There are times when it is important that your application know what the user has done with STSTray so that an action can be taken. For example, if the user selects from a menu item created with `create menu`, or clicks a particular button in an `answer` dialog box[4], or double-clicks the icon, there needs to be some way for STSTray to communicate with your application.

To do this, STSTray writes out a text file that your application can be looking for in a specific directory. When the file is written by STSTray, your application can pick it up and act on it.

STSTray will automatically send data to an "output file" called `output.txt` that resides in the same directory as the STSTray application. This can be changed using the `set outputFile` command. When STSTray writes data to its output file it will automatically create the file if it does not exist, or will append to it if it exists. It is the responsibility of your application to delete or rename this file if you don't want this appending to occur (or to use the `delete file` command in STSTray).

[4] This assumes you chose not to implement the `switch result` block (see the `answer` command for more info).

Special Characters

Many commands have text parameters. For those commands, you can use the following special characters, which will be substituted automatically during the command's operation:

\n	new line (linefeed/return)
\'	Double quotation mark (")
%d%	Inserts the current date
%t%	Inserts the current time

For example, the following command:

```
answer "This is line 1\nThis is line 2"
```

would bring up a message box that showed this:

```
This is line 1
This is line 2
```

File Paths

Many commands accept file paths as parameters. In those cases, you can either supply an absolute path or a relative one (both \-delimited and /-delimited are accepted), and if the path is relative, you can use ../ or ..\ variations to "back up" in the directory hierarchy. File paths are not case sensitive. Relative paths are calculated based on the location of the currently running STSTray executable. So for example, if STSTray were located at C:\Program Files\STSTray\STSTray.exe, and you wanted to refer to a file called "MyFile.txt" that was at the root of the C: drive, any of the following paths will work and are equivalent:

```
C:\MyFile.txt
C:/MyFile.txt
../../MyFile.txt
..\..\MyFile.txt
```

Quitting STSTray

STSTray is designed to attempt to prevent you from leaving an icon in the tray and having no way to remove it. So there are three ways to quit STSTray (thus removing the icon from the system tray):

1. If you have not assigned a menu to the tray icon with the `create menu` command, clicking the icon with the left mouse button will quit STSTray.
2. If you *have* assigned a menu to the tray icon with the `create menu` command, and the menu has a "Quit" or "Exit" menu item, selecting either "Quit" or "Exit" will quit STSTray, *unless* you have defined a `case` statement in the `itemSelect` handler to trap the "Quit" or "Exit" menu items and execute your own custom commands. (NOTE: If you do choose to trap either "Quit" or "Exit" yourself, please make sure to include the `quit` command at the end of your custom commands, otherwise STSTray will not be able to be closed through a mouse or menu action.)
3. Executing the `quit` command from an `on openScript` handler in a drop script and putting it in the same directory as STSTray (so that it loads and executes).

Commands

answer

Summary

This command displays an alert dialog box, with an icon, title, and your choice of buttons.

Syntax

There are actually two syntax forms for this command, depending whether you want the result (the button the user clicked) to be written to the output file or whether you want to take action on the choice directly in the script. The basic syntax is this:

```
answer [{info[rmation]|question|warning|error}] text
  [with {"OK"|"OKCancel"|"YesNo"|"YesNoCancel"}] [titled title]
```

If you want the result to be written to the output file, this is all you need to do. If, however, you want to trap the result and take action on it in the script, you need to follow the `answer` command with the following `switch result` construct:

```
switch result
  case btn1Name
    commands
  [case btn2Name
    commands
  [case btn3Name
    commands]]
end switch
```

Note that even if there is only one button, you still need to use `switch result` in order to execute commands after the user has clicked the button.

Arguments

`info[rmation]`
`question`
`warning`
`error`
	Optional. This is the type of icon to display. If no icon type is chosen, STSTray will use the information icon.

`text`	This is the text to display in the dialog box, enclosed in quotes. Special characters are substituted (see Special Characters, above).

`OK`
`OKCancel`
`YesNo`
`YesNoCancel`
	Optional. These are the kinds of buttons that the user can choose from. When a button is selected, the chosen button is sent to the output file (if there is no `switch result` construct following the `answer` command), or will attempt to be matched to a case in a `switch result` construct (if one exists). Note that these are case-sensitive. If not provided or incorrectly provided, it will default to "OK".

`title`	Optional. This is the title to display in the dialog box, enclosed in quotes. If not specified, the title bar will be empty. Special characters are substituted (see Special Characters, above).
`btn1Name,` *`btn2Name,`*	Used by the `switch result` construct, matches to the specific button clicked will execute the indented commands below the `case` statement.

btn3name

Description

This command will bring up a dialog box based on the arguments above. When a button is clicked, one of two things will happen:

1. If a `switch result` construct appears immediately after the `answer` command, an attempt will be made to match the name of the button clicked with a corresponding `case` statement in the `switch result` construct. If a match is found, the commands listed beneath that `case` statement are executed. If no match is found, nothing will happen and no error will be generated. Note that there is no `case` for default action if nothing is matched; you need to provide a `case` statement for every button choice that you wish to take action on.

   If a button takes no action (for example, clicking the "No" choice in a "Are you sure you want to delete this file?" dialog box), you have three options: (a) you can omit the `case` completely, (b) you can include the `case`, but not provide any statements underneath the `case`, or (c) you can include the `case` and insert the `do nothing` command underneath the `case`. See the entry for the `do nothing` command for more info.

2. If there is no `switch result` construct immediately below the `answer` command, the name of the button will be written to the output file (see The "Output File" above for more information).

Example 1:

This script displays this dialog box when the script file is opened, and writes the result to the output file:


```
on openScript
  answer info "Are you happy today?" with "YesNo" titled "Happy Poll"
end openScript
```

Example 2:

This script does the same thing as Example 1, but provides feedback to the user based on their response:

```
on openScript
  answer info "Are you happy today?" with "YesNo" titled "Happy Poll"
  switch result
    case "Yes"
      answer "Glad to hear it!"
    case "No"
      answer "I'm sorry to hear that!"
  end switch
end openScript
```

Note the response to the feedback dialog box (which would display a simple "OK" button) are written to the output file since there is no `switch result` construct that exists underneath the feedback `answer` commands.

See also

do nothing, set outputFile, write

beep

Summary

This command simply issues a beep sound.

Syntax

```
beep
```

Arguments

none

Example

This is an example of causing a beep to sound when a menu item called “Play Beep” is selected:

```
on itemSelect
  case "Play Beep"
    beep
  -- of course there would be more menu items defined here
end itemSelect
```

check for updates

Summary

This command is used to automatically download and examine a drop script file from a web server, and if it is newer than the last time it checked, it will execute the commands within that drop script file. This is usually used to notify a number of users simultaneously of a product update, or other piece of information.

Syntax

```
check url [for updates]
```

Arguments

url	The absolute URL to the drop script file on the web server.

Description

This command, when executed, will download the drop script file located at *url*, open it, and see if it is newer, and if it is, it will run it as if the file had been dropped into the STSTray directory (see Polling Mode above, for more information on drop scripts).

To do this, STSTray needs to have a method of identifying whether a drop script is newer or not. It does this by comparing *script IDs*, which are set specifically using the `set scriptID` command inside the `on openScript` handler of the drop script you are downloading from *url*. Here’s how it works:

- When STSTray launches, it checks to see if there is a `ststray.ini` file in the same directory as the STSTray executable. The format of this INI file is very simple, and looks something like this:

  ```
  [Updates]
  LastID=1000
  ```

- If the `ststray.ini` file exists, it looks into it and tries to read the `LastID` key of the `[Updates]` section. If then sets an internal `lastID` variable to the ID number it reads from the INI file (which would be 1000 in the example above).

- If the `ststray.ini` file does not exist, or it exists but does not have the proper `LastID` key of the `[Updates]` section, it creates the file with an empty `LastID` key and sets the internal `lastID` variable to empty.
- When the `check for updates` command is executed, it downloads the drop script file at *url* to a temporary file in the same directory as the STSTray executable (to a file called `ststray.tmp`), opens it up, examines its `on openScript` handler, and looks for a `set scriptID` command.
    - If it finds one, it extracts the ID from the `set scriptID` command (the "incoming ID") and compares it against the value stored in its `lastID` internal variable (the "stored ID").
        - If the incoming ID is the same as the stored ID, it means that this script has already been read and executed before, so it does nothing, and deletes the temporary file.
        - If the incoming ID is different than the stored ID, it will write the incoming ID to the `LastID` key in the INI file, and set its stored ID to the value of the incoming ID. It will then execute the script file as if it had been dropped in the STSTray directory (running the commands contained in the file's `on openScript` handler (if any)), and finally delete the temporary file.
    - If it doesn't find a `set scriptID` command in the temporary drop script file, it means the drop script file was not set up properly, so it ignores it and deletes the temporary file.
- If for some reason no file can be found at *url*, no error is generated. The reason for this is that is provides more flexible deployment support (see the Example below).

The `check for updates` command can be executed manually (i.e. attached to a menu item or an `on openScript` handler), or for maximum utility, it can be executed periodically using the `set timer` command and the `on timerEvent` handler, as shown in Example 1 below.

You may want to know whether there was an update or not, especially if `check for updates` was executed manually. To do this, STSTray can report whether the incoming ID is the same as the stored ID and you can take action on this by following the `check for updates` command with the `switch result` construct:

```
switch result
  case "same"
    commands
  case "different"
    commands
end switch
```

This `switch result` is evaluated *after* any downloaded drop script is executed. Note that just as in `answer`, you can choose to omit either of the `cases` above or use the `do nothing` command if you want to show both cases but take no action on one of them. See Example 2 below for how this might be used.

Example 1:

You have a product that you sell that you want to be able to notify users immediately when a new update is available. So you include STSTray in your installer, install it in a directory of your choosing, and install a shortcut to STSTray in your Startup folder to make sure that it loads properly every time the user starts up their computer. You install a boot script that will check for updates every 30 minutes:

```
on openScript
  set the icon to "icons/myicon.gif"  -- installs custom icon
  set the timer to 30 minutes
end openScript

on timerEvent
  check "http://www.mycompany.com/updates/updates.scp" for updates
end timerEvent
```

STSTray by default installs with no `ststray.ini` file. Every 30 minutes, STSTray will attempt to download and open the drop script file at `http://www.mycompany.com/updates/updates.scp`. Since you haven't uploaded the `updates.scp` file yet (there's no need to notify the user of anything since they just bought and installed your product!), nothing happens when the file can't be found, and it waits another 30 minutes before checking again.

Months later, you release an update to your product that you want to notify your users about. So you create an update file called `updates.scp` and since you know you've never sent out an update notification before, you can set your scriptID to anything and it will be considered "newer" by STSTray. So you create a script like this and uploaded it to your web server at `http://www.mycompany.com/updates/updates.scp`:

```
on openScript
  set the scriptID to 1000
  show info balloon "A new version of MyProduct is available. Click here to see
what's new." titled "New Version of MyProduct"
end openScript

on balloonClick
  -- Go to a web page that gives information on the new version
  goURL "http://www.mycompany.com/products/MyProductV2.htm"
end balloonClick

on altShowBalloon
  flash icon
  install handler:
    on clickOneTime
      stop flashing
      answer info "A new version of MyProduct is available. Would you like
to see what's new?" with "YesNo" titled "New version of MyProduct"
        switch result
          case "Yes"
            goURL "http://www.mycompany.com/products/MyProductV2.htm"
        end switch
    end clickOneTime
end altShowBalloon
```

The next time STSTray on the user's machine checks for updates, it will see the file is there, download it, get the `scriptID` (1000) and check it against what it previously had stored (nothing). It is obviously new, so it sets it stored ID to 1000, creating the `ststray.ini` file (since there wasn't one there before) and setting the `LastID` key in the INI file to 1000, and loads the script. It executes the `show balloon` command, which will display the balloon on Windows XP or greater – if they click on the balloon, it will take them to the web page with more information about the product update. If they don't have Windows XP or greater, the `on altShowBalloon` handler will kick in and it will flash the icon in the tray, and install a one-time handler that will activate when the user clicks the icon with their mouse. When they click, it will execute the commands in the installed `on clickOneTime` handler, and will bring up an `answer` dialog box, letting them know a new version of your product is available. If they click "Yes", they will be taken to the web page with more information on the product update. If they click "No", the dialog box goes away. In either case, STSTray will go back to checking for updates every 30 minutes.

Example 2:

This is the same as Example 1, but instead of an automated check, you have added a "Check for Updates" item in your menu item so that they can check manually. In this case, you want to let them know whether or not an update is available. So you could do this:

```
on itemselect
  case "Check for Updates"
        check "http://www.mycompany.com/updates/updates.scp" for updates
```

```
        switch result
               case "same"
                      answer "You are using the current version of MyApp."
               case "different"
                      do nothing
        end switch
end itemselect
```

Note: The reason that nothing is executed in the "different" case is because the check for updates command would have downloaded a drop script that executed and taken whatever action(s) were required. In actuality, most people won't use the "different" case for this reason, but *will* take advantage of the "same" case.

See also

set timer, on timerEvent [handler], getURL, download, do nothing

create menu

Summary

This command creates a custom menu that will be displayed when the `popup menu` command is executed.

Syntax

`create menu "`*`item1,item2,…,itemN`*`"`

Arguments

`item1 … itemN`	These are the names of menu items to display in the menu, enclosed in quotes. The order they are displayed is from top-to-bottom (i.e. *item1* is at the top of the menu, and *itemN* is at the bottom). You can insert a separator line by making one of the items a hyphen ("-"). You can make a menu item disabled by preceding it with an open parenthesis ("(") Note that the entire list of menu items is quoted; **do not** enclose each menu item in quotes, or you will only see the first one.

Description

This command will install a menu into the system tray icon that you can display using the `popup menu` command.

When a menu item is selected, STSTray looks for an `on itemSelect` handler in the Active Handlers for the selected menu item. If it finds one, it executes the commands inside that item's `case` section of the script (see `on itemSelect` under Handlers for more info).

If it does not find a corresponding `on itemSelect` handler for the selected menu item, the name of the menu item is written to the output file (see The "Output File" for information on output files).

Any menu item named "Quit" or "Exit" will automatically be handled by STSTray, and will cause STSTray to quit and remove itself from the system tray, if it is not trapped by a `case` in `on itemselect`.

Note that this command will replace any previously installed menu with the menu being created; this is a good way to create dynamic menus.

Example

This script creates a menu with three items. Selecting "Greeting" causes a message box to appear with a greeting, selecting "Exit" will close STSTray.

```
on openScript
  create menu "Greeting,-,Exit"
end openScript

on itemSelect
  case "Greeting"
    answer information "Hello there!" with "OK" titled "Greeting"
end itemSelect
```

See also

popup menu, set outputFile, on itemSelect [Handler]

delete file

Summary

This command deletes a local file.

Syntax

`delete [file] filePath`

Arguments

filePath	This is the path to the file that is to be deleted, enclosed in quotes. (See File Paths for file path specification options.)

Example

The following example deletes the file `deleteme.txt` that is located at the root level of the C:\ drive when the script file is opened:

```
on openScript
  delete file "c:\deleteme.txt"
end openScript
```

See also

write, set output

do nothing

Summary

This command does absolutely nothing, and is usually used by those who wish to show all `cases` in an `answer` or `check for updates` command. You can also use the simple string `nothing` instead of `do nothing` if you prefer.

Syntax

`do nothing`

Arguments

none

Example

The following example shows how this might be used

```
on openScript
  answer "Are you sure you want to launch Notepad?" with "YesNo"
  switch result
    case "Yes"
      launch "C:\WINDOWS\NOTEPAD.EXE"
    case "No"
      do nothing
  end switch
end openScript
```

See also

answer

download

Summary

This command downloads a specified file from a web server to the local drive. Note that commands that follow this command will not be executed until the download has been completed.

Syntax

`download` *`url`* [`to` *`filePath`*]

Arguments

`url`	The absolute URL of the file you wish to download.
`filePath`	Optional. This is the path to the place where the file is to be retrieved, enclosed in quotes, and including the name of the file after it has been retrieved. (See File Paths for file path specification options.) If you do not provide a *filePath*, the file at *url* will be downloaded to the STSTray folder, and will be named the same as it was on the web server.

Example 1:

This example downloads a file at `http://www.mycompany.com/testfile.htm` to the STSTray directory (depositing a file called `testfile.htm` in that directory) when the user double-clicks on the tray icon:

```
on doubleClick
  download "http://www.mycompany.com/testfile.htm"
end doubleClick
```

Example 2:

This is the same as Example 1, but downloads the file to the `C:` drive and changing its name to `newfile.htm`:

```
on doubleClick
  download "http://www.mycompany.com/testfile.htm" to "C:\newfile.htm"
end doubleClick
```

See also

goURL, check for updates

flash icon

Summary

This command causes an icon to be loaded and flash in the tray.

Syntax

```
flash icon [iconPath] [every num [milliseconds|ms]]
```

Arguments

iconPath	Optional. This is the path to the icon, enclosed in quotes. (See File Paths for file path specification options.) The "icon" can be a GIF, JPEG, BMP or ICO file, and if it is larger than 16x16 it is automatically scaled to fit. If no icon is specified, executing this command will cause the currently loaded icon in the system tray to flash.
num	Optional. The number of milliseconds between flashes; defaults to 500 milliseconds if not specified. If specified, must be an integer between 1 and 65535.

Description

Calling this command will load and flash the icon specified in *iconPath* every *num* milliseconds. It will replace the currently displayed icon (if there is one).

Example

The following example flashes the icon `myicon.jpg` that resides in the subfolder `icons` every 750 milliseconds:

```
on openScript
  flash icon "icons/myicon.jpg" every 750 ms
end openScript
```

See also

set icon, swap icon, show balloon, stop flashing, stop swapping

goURL

Summary

This command executes an internet URL.

Syntax

```
goURL url
```

Arguments

url	The absolute URL that you wish to execute.

Description

This command can be used to open or go to the user's web browser and navigate to a web site (with an "http" URL), create a new mail message using the user's local email client (with a "mailto:" URL), or any other type of URL that you could execute in a web browser.

Example

The following example goes to a web page on a web site when the user picks the right menu item from the popup menu that is displayed.:

```
on itemSelect
  case "Go to My Home Page"
    goURL "http://www.mycompany.com/products/"
end openScript
```

See also

download, check for updates

install handler

Summary

This command installs a new handler on demand.

Syntax

There are two forms of syntax for this command:

```
install [handler] handlerText
```

```
install [handler]:
  handler
```

Arguments

handlerText	This is the text of the handler as a quoted string that uses \n for new lines and \' for embedded quotation marks (see Special Characters, above). The *handlerText* needs to resolve to a real handler (see the examples).
handler	This is a non-quoted handler that is multiple lines and appears like a normal handler would; it's just that it is within another handler (see the examples).

Description

This command provides the ability to install a new handler on the fly, as if that handler were in a drop script that STSTray had processed. It can be very useful to install a new handler (especially a one-time handler) based on a menu selection (see `on itemSelect`) or button selection in an alert dialog (see `answer`), or as the result of checking for updates (see `check for updates`).

Note that if using the second form (the more expanded and readable form), you need to make sure you end the `install handler` command with a colon (":") otherwise STSTray won't know that the next lines are a handler, and it will try to process them as commands and generate script errors.

Example 1:

This example replaces the existing `leftClick` handler with a new one based on a menu selection:

```
on itemSelect
  case "Replace leftClick"
    install handler "on leftClick\nanswer \'Hello\'\nend leftClick"
end itemSelect
```

After the menu item has been chosen, clicking on the icon in the tray will display an alert box that says "Hello".

Example 2:

This is the same example, but using the expanded (more readable) form:

```
on itemSelect
```

```
  case "Replace leftClick"
    install:
      on leftClick
        answer "Hello"
      end leftClick
end itemSelect
```

Please see the Example in the `check for updates` entry for another example of how this command can be used.

See also

answer, on itemSelect [Handler], check for updates

launch

Summary

This command launches an application or document, optionally with additional parameters

Syntax

`launch` ***`filePath`*** `[with` ***`parameters`***`]`

Arguments

`filePath`	This is the path to the application or document, enclosed in quotes. (See File Paths for file path specification options.)
`parameters`	This is any parameters that the application needs, enclosed in quotes . (See File Paths for file path specification options.)

Example 1:

This script launches the Notepad application, or a Word document:

```
on openScript
  create menu "Open Notepad,Open Word Doc,-,Quit "
end openScript

on itemSelect
  case "Open Notepad"
    launch "c:\windows\notepad.exe"
  case "Open Word Doc"
    launch "c:\myDocument.doc"
end itemSelect
```

Example 2:

This script launches a specific Notepad document using parameters:

```
on openScript
  create menu "Open Notepad File,-,Quit "
end openScript

on itemSelect
  case "Open Notepad File"
    launch "c:\windows\notepad.exe" with "c:\myfile.txt”
end itemSelect
```

popup menu

Summary

This command pops up the menu created with `create menu`.

Syntax

```
popup menu
```

Arguments

none

Description

Normally, a menu created with `create menu` will pop up automatically whenever the user right-clicks on the tray icon. You might wish to pop the menu up at other occasions; for example, when the user left-clicks, or double-clicks on the tray icon.

`popup menu` will not work if a menu has not previously been created with `create menu`.

Example:

Here is an example of popping up the menu when the left mouse button is clicked (as well as the default right-mouse button):

```
on click
  popup menu
end click
```

See also

create menu, on itemSelect [Handler]

quit

Summary

This command quits STSTray and removes its icon from the system tray.

Syntax

```
quit
```

Arguments

none

Description

There are a number of built-in ways to quit STSTray without having to use the `quit` command (see Quitting STSTray for a list of these approaches), and most of the time will not need to be used. However, in certain instances it can be useful (see the example below).

Note that `exit` can be used in place of `quit`.

Example

This script sets a custom Quit menu item:

```
on openScript
  create menu "Greetings,Howdy,-,Quit MyApp"
```

```
end openScript

on itemSelect
  case "Quit MyApp"
    quit
end itemSelect
```

set icon

Summary

This command sets the icon to be used in the system tray.

Syntax

`set [the] icon to` *`iconPath`*

Arguments

`iconPath`	This is the path to the icon, enclosed in quotes. (See File Paths for file path specification options.) The "icon" can be a GIF, JPEG, BMP or ICO file, and if it is larger than 16x16 it is automatically scaled to fit.

Description

Calling this command will change the icon in the tray from what is currently being displayed to the icon provided in *iconPath*. If the icon is currently flashing (see the `flash icon` command), it will be replaced by the icon in *iconPath* and the flashing will continue. If the icon is currently being swapped (see the `swap icon` command), the icon provided in *iconPath* will replace the "main" icon, and not the "swap" icon (see the entry on the `swap icon` command for more information), and the swapping will continue with the new icon.

Example

The following example changes the existing icon to the icon `myicon.gif` in the subfolder `icons`:

```
on openScript
  set the icon to "icons/myicon.gif"
end openScript
```

See also

flash icon, swap icon

set outputFile

Summary

This command defines the location of the output file.

Syntax

`set [the] outputFile to` *`filePath`*

Arguments

`filePath`	This is the path to the output file, enclosed in quotes. (See File Paths for file path specification options.)

Description

Normally, commands like `answer` and selected menu items without scripts will output their selections to a file called `output.txt` located in the same directory as the STSTray application (see The "Output File" for more information). You can change this location using the `set outputFile` command so you can direct output to wherever you need it.

Example

This script outputs the selected menu item to a user-defined output file:

```
on openScript
  set outputFile to "myoutput/menuselections.txt"
  create menu "Greetings,Hello,Salutations,-,Exit"
end openScript
```

See also

answer, write

set polling

Summary

This command sets the frequency that STSTray looks for drop scripts.

Syntax

`Set [the] polling to` *`num`* `[{milliseconds|ms}]`

Arguments

`num`	The number of milliseconds between each polling attempt. Normally STSTray polls for drop scripts every 500 milliseconds; this overrides the default polling interval. Must be an integer between 0 and 65535; if set to 0, STSTray will never look for drop scripts.

Description

Depending on your needs, you may wish to set the `polling` to higher than the default. This will reduce the CPU activity on the user's computer, and will help to prevent interference with things that require a lot of performance (such as video playback).

You may turn off polling completely by setting the `polling` to 0, but that means that STSTray will no longer look for drop scripts. The only way to restore the polling capability is to change the `polling` value via a menu item, or via a reboot (unless you set the `polling` to 0 in the `on openScript` handler in the boot script), or via a script file that is retrieved with `check for updates` (which is processed regardless of the polling setting, as long as it is newer (see `check for updates` for more information)).

Example

The following example turns off polling when STSTray launches, but allows it to be reset to the default when the proper menu item is selected:

```
on openScript
  set the polling to 0
  create menu "Reset Polling,-,Exit"
end openScript

on itemSelect
```

```
    case "Reset Polling"
      set the polling to 500
  end itemSelect
```

See also

set timer

set scriptID

Summary

Defines a unique identifier for a drop script file, used by `check for updates` to see if the drop script needs to be processed.

Syntax

```
set [the] scriptID to value
```

Arguments

value	Any string or numeric value.

Description

This command sets a unique identifier for a particular script. It is used only by the `check for updates` command; for more information and an example of how it is used, refer to the `check for updates` command.

See also

check for updates

set silentErrors

Summary

Determines whether errors generated by STSTray are displayed as alert boxes to the screen, or written to an error log file.

Syntax

```
set [the] silentErrors to {true|false}
```

Arguments

none

Description

Normally, errors that are generated by STSTray are reported by displaying a dialog box on the screen. This can be a bit embarrassing for end users to see, so there is an option to redirect errors to a log file, or to handle them yourself via the `on scriptError` handler.

When STSTray launches, the `silentErrors` are automatically set to false; if you set it to true and an error is generated, STSTray will check to see if there is an `on scriptError` handler that is loaded in the Active Handlers. If there is, it will trigger that handler and execute whatever commands are inside that handler. If not, errors will be written to a file called `errors.log`, which is in the same directory as the STSTray executable. If the file already exists it will be appended to; if it does not exist, it will be created.

Example

The following example turns off the display of STSTray's error dialogs, and provides a custom error dialog instead (using the on scriptError handler).

```
on openScript
  set the silentErrors to true
end openScript

on scriptError
  answer error "An error was generated performing the last action. Please
   contact MyCompany at 555-1212 and report the error." titled "Error
   Encountered"
end scriptError
```

See also

on scriptError [Handler]

set timer

Summary

This command sets up a countdown timer that will execute the commands in the on timerEvent handler when the time expires.

Syntax

```
set [the] timer {to|for} num [{ms|milliseconds|sec|secs|second|seconds|min|mins
 |minutes|hr|hrs|hours}]
```

Arguments

num	The number of milliseconds, seconds, minutes or hours to set the countdown timer for. If a timer is already active, setting the timer to 0 will turn off the timer.

Description

This command is used to set up the timer so that you can do timed commands. When the timer interval has passed, it executes the commands in the on timerEvent handler (if loaded). If for some reason a timer has been set, but no on timerEvent handler has been loaded, nothing will happen when the timer interval has passed.

STSTray has only one timer, so if you want to stop and restart the timer, you can execute stop timer or set the timer to 0 and then set it again to a new value.

Note: One thing to consider when using set timer is that the timer interval starts as soon as the set timer command is executed. This means that if you set the timer to something like 2 days, and the set timer command is executed in a boot script, but the user shuts down their computer every night and reboots the next morning, your timer will never trigger. It is therefore suggested that you keep your timers to 8 hours or less unless you know the machine is going to stay on for a long time.

Example

The following is an example that displays an alert box each time an hour has passed:

```
on openScript
  set the timer to 1 hour
end openScript

on timerEvent
  answer "An hour has passed."
```

```
end timerEvent
```

Another example can be seen in the Example section of the `check for updates` command.

See also

stop timer, check for updates, on timerEvent [Handler]

set tooltip

Summary

This command defines the tooltip to be displayed when the mouse passes over the icon in the tray.

Syntax

`set [the] tooltip to` *`text`*

Arguments

`text`	The text of the tooltip, enclosed in quotes. Special characters are substituted (see Special Characters, above). *Note: If the target machine is running Shell 5 (that is, Internet Explorer 5) or higher, the maximum number of characters that can be displayed is 127. If the target machine is running a Shell that is less than 5 (Internet Explorer 4 for example), the maximum number of characters that can be displayed is 63. If you supply more than the maximum allowable characters, it will be truncated.*

Description

If you do not specify a tooltip, STSTray defaults to showing a tooltip that says “STSTray”.

Example

This script sets the tooltip to “MyApp – Paused”

```
on openScript
  set tooltip to "MyApp – Paused"
end openScript
```

See also

show balloon

show balloon

Summary

This command causes a balloon to be displayed under Windows XP or greater.

Syntax

`show [{info[rmation]|warning|error}] balloon with` *`text`* `titled` *`title`*

Arguments

`info[rmation]`
`warning`
`error`
	Optional. These are the icon types that can be displayed next to the title of the balloon:

If no icon type is defined, the “information” icon is used by default.

text — The text message that should be displayed when the balloon is displayed, enclosed in quotes. Special characters are substituted (see Special Characters, above).

Note: If the target machine is running Shell 5 (that is, Internet Explorer 5) or higher, the maximum number of characters that can be displayed is 255. If the target machine is running a Shell that is less than 5 (Internet Explorer 4 for example), the maximum number of characters that can be displayed is 127. If you supply more than the maximum allowable characters, it will be truncated.

title — The title in the balloon to display when the balloon is displayed, enclosed in quotes. Special characters are substituted (see Special Characters, above).

Note: If the target machine is running Shell 5 or higher, the maximum number of characters that can be displayed is 127. If the target machine is running a Shell that is less than 5, the maximum number of characters that can be displayed is 63. If you supply more than the maximum allowable characters, it will be truncated.

Description

This command will display a balloon in the system tray based on the parameters provided above. This is the kind of command that is usually not included in the boot script file that loads when STSTray starts up; it is generally implemented as a drop script later, when a specific event occurs. It will also generate a `balloonShow` message which will execute any commands in the `on balloonShow` handler (if loaded).

Note: This command only works under Windows XP or greater; if used under another environment, it will generate an `altShowBalloon` message which will execute any commands in the `on altShowBalloon` handler (if loaded).

If the user clicks anywhere on the balloon itself (other than the close box), it will generate a `balloonClick` message and execute any commands in the `on balloonClick` handler (if loaded). If the close box is clicked, it will generate a `balloonClose` message and execute any commands in the `on balloonClose` handler (if loaded).

Example

The following example displays the Information balloon shown in the graphic above:

```
on openScript
  show information balloon with "Isn't this cool?" titled "STSTray"
end openScript
```

See also

flash icon, swap icon, on altShowBalloon [Handler], on balloonClick [Handler], on balloonClose [Handler], on balloonShow [Handler]

stop flashing

Summary

This command causes the icon in the tray to stop flashing or swapping, returning it to its original icon. If no icon is flashing or swapping, it will have no effect. Another form of this command is `stop swapping`.

Syntax

```
stop flash[ing]
```

Arguments

none

Example

To see a good example of this, take a look at the Example in the `check for updates` command.

See also

flash icon, swap icon, stop swapping

stop swapping

Summary

This command causes the icon in the tray to stop flashing or swapping, returning it to its original icon. If no icon is flashing or swapping, it will have no effect. Another form of this command is `stop flashing`.

Syntax

```
stop swap[ping]
```

Arguments

none

Example

To see a good example of this, take a look at the Example in the `check for updates` command.

See also

flash icon, swap icon, stop flashing

stop timer

Summary

This command causes the currently running timer to stop and be reset, allowing you to change the timer settings using `set timer`, or just leave it off. Another form of this command is `set the timer to 0`.

Syntax

```
stop timer
```

Arguments

none

Example

To see a good example of this, take a look at the boot.scp script that comes with STSTray.

See also

set timer

swap icon

Summary

This command animates two icons (swapping them back and forth) in the system tray.

Syntax

```
swap icon[s] mainIconPath {and|with} swapIconPath [every num [milliseconds|ms]]
```

Arguments

mainIconPath	This is the path to the "main" icon, enclosed in quotes. (See File Paths for file path specification options.) The "icon" can be a GIF, JPEG, BMP or ICO file, and if it is larger than 16x16 it is automatically scaled to fit.
swapIconPath	This is the path to the "swap" icon, enclosed in quotes. Same rules as for *mainIconPath* apply.
num	Optional. The number of milliseconds between swaps; defaults to 500 milliseconds if not specified. Must be an integer between 1 and 65535.

Description

This command is similar to `flash icon`, except that instead of one icon blinking, you have two icons that trade places over and over again. If an icon already exists, it will be replaced.

Example

This example swaps a mailbox icon (mail1.ico) with another mailbox icon with the flag up (mail2.ico). Both icons reside in the directory *above* the STSTray application, using the default interval (500 ms).

```
on openScript
  swap icons "../mail1.ico" and "../mail2.ico"
end openScript
```

See also

set icon, flash icon, show balloon

write

Summary

This command writes a chunk of text to a user-specified text file, or to the output file if a path is not specified.

Syntax

```
write text [to file filePath]
```

Arguments

text	This is the text to write, enclosed in quotes. Special characters are substituted (see Special Characters, above).
filePath	Optional. This is the path to the file where the text should be written, enclosed in quotes. (See File Paths for file path specification options.)

Description

The `write` command can be used to write a user-defined piece of text to a user-defined file. Compare this with `set outputFile`, which just redirects "standard" output to a text file. Note that you can use the special characters %d% and %t% to write the date and time to the file you create. This makes it quite useful for creating logs of activity.

Example

This script outputs a custom chunk of text based on each selected menu item (see the entry on `itemSelect` for more information on how `itemSelect` works):

```
on openScript
  create menu "Greetings,Hello,-,Exit"
end openScript

on itemSelect
  case "Greetings"
    -- This goes to the standard output file
    write "Greetings from your friend, Max!"

  case "Hello"
    -- This goes to a specified file path
    write "Hello everyone!" to file "greetings.txt"
end itemSelect
```

See also

answer, set outputFile

Handlers

on altShowBalloon

Summary

This handler is executed whenever STSTray is supposed to display a balloon (done via the `show balloon` command) but the current operating system is not Windows XP or greater.

Syntax

```
on altShowBalloon
  commands
end altShowBalloon
```

Description

This handler can be used to set up a universal notification approach, regardless of operating system. You can execute a `show balloon` command and know that if the user isn't running Windows XP or greater that it will generate the `altShowBalloon` event that can be trapped here, and you can do something else for non-XP systems.

Example

The following example flashes the icon in the tray, and then when clicked (once only), it stops the flashing of the icon and brings up an `answer` dialog box.

```
on altShowBalloon
```

```
  flash icon
  install handler:
    on clickOneTime
      stop flash
      answer info "A new version of STSTray is available." with "OK" titled
"New Version”
    end clickOneTime
end altShowBalloon
```

See also

show balloon

on balloonClick

Summary

This handler is executed whenever the user clicks on a balloon that was displayed with the `show balloon` command, anywhere but on the close box of the balloon.

Syntax

```
on balloonClick
  commands
end balloonClick
```

Description

This is quite frequently used to let the user know that clicking on the balloon will do something like “click the balloon to go to our web site” or “click the balloon to start the installation”.

Example

The following handler is used to take the user to a web site when they click on it:

```
on balloonClick
  goURL "http://www.sonsothunder.com/"
end balloonClick
```

See also

show balloon, on balloonClose [handler], on balloonShow [handler]

on balloonClose

Summary

This handler is executed whenever the user clicks the close box of a balloon displayed with the `show balloon` command.

Syntax

```
on balloonClose
  commands
end balloonClose
```

Description

This is very infrequently used (most of the time you don't care if the balloon is closed or not), but you *might* want to take some clean-up action if the user chooses to not acknowledge the balloon alert by clicking on it.

Example

Here's an example that assumes that a file was downloaded containing some "read me" information that would be displayed if the user clicked on the balloon (the message said "Click here to read what is new in 2.0."). Clicking the close box deletes the file (since the user doesn't want to see it right now).

```
on balloonClose
  delete file "20ReadMe.txt"
end balloonClose
```

See also

show balloon, on balloonClick [handler], on balloonShow [handler]

on balloonShow

Summary

This handler is executed whenever the balloon is displayed with the `show balloon` command.

Syntax

```
on balloonShow
  commands
end balloonShow
```

Description

You may wish to take certain actions when the balloon is shown; this handler will allow you to do that.

Example

Here's an example of handler that writes a log file noting that the balloon was seen and when it was seen.

```
on balloonShow
  write "Balloon was displayed at %d% %t%." to file "balloonlog.txt"
end balloonShow
```

See also

show balloon, on balloonClick [handler], on balloonClose [handler]

on doubleLeftClick (on doubleClick)

Summary

This handler is executed whenever the user double-clicks the icon in the system tray with the left mouse button.

Syntax

```
on doubleLeftClick
  commands
end doubleLeftClick

on doubleClick
  commands
end doubleClick
```

Example

This is an example where double-clicking the icon with the left mouse button will bring up an `answer` dialog:

```
on doubleClick
  answer "You double-clicked me."
end doubleClick
```

See also

on leftClick [handler], on leftClickOneTime [handler], on doubleLeftClickOneTime [handler], on rightClick [handler], on rightClickOneTime [handler], on doubleRightClick [handler], on doubleRightClickOneTime [handler]

on doubleLeftClickOneTime (on doubleClickOneTime)

Summary

This handler is executed the first time a user double-clicks the icon in the system tray with the left mouse button.

Syntax

```
on doubleLeftClickOneTime
  commands
end doubleLeftClickOneTime
```

```
on doubleClickOneTime
  commands
end doubleClickOneTime
```

Description

The first time the user double-clicks the icon in the system tray, this code will execute. The second and subsequent times the user double-clicks the icon, the `on doubleLeftClick` handler will be executed. Note that you can "reinstall" a one-time handler by adding it again in via another drop script file.

Example

The following example displays a dialog box only the first time the icon is double-clicked:

```
on doubleLeftClickOneTime
  answer "You double-clicked me! But you won't see this again..."
end doubleLeftClickOneTime
```

See also

on leftClick [handler], on leftClickOneTime [handler], on doubleLeftClick [handler], on rightClick [handler], on rightClickOneTime [handler], on doubleRightClick [handler], on doubleRightClickOneTime [handler]

on doubleRightClick

Summary

This handler is executed whenever the user double-clicks the icon in the system tray with the right mouse button.

Syntax

```
on doubleRightClick
  commands
end doubleRightClick
```

Example

This is an example where double-clicking the icon with the right mouse button will write text to the output file:

```
on doubleRightClick
  write "I was double-clicked."
end doubleRightClick
```

See also

on leftClick [handler], on leftClickOneTime [handler], on doubleLeftClick [handler], on doubleLeftClickOneTime [handler], on rightClick [handler], on rightClickOneTime [handler], on doubleRightClickOneTime [handler]

on doubleRightClickOneTime

Summary

This handler is executed the first time a user double-clicks the icon in the system tray with the right mouse button.

Syntax

```
on doubleRightClickOneTime
  commands
end doubleRightClickOneTime
```

Description

The first time the user double-clicks the icon in the system tray with the right mouse button, this code will execute. The second and subsequent times the user double-clicks the icon, the `on doubleRightClick` handler will be executed. Note that you can "reinstall" a one-time handler by adding it again via another drop script file.

Example

The following example displays a dialog box only the first time the icon is double-clicked with the right mouse button:

```
on doubleRightClickOneTime
  answer "You double-clicked me! But you won't see this again..."
end doubleRightClickOneTime
```

See also

on leftClick [handler], on leftClickOneTime [handler], on doubleLeftClick [handler], on doubleLeftClickOneTime [handler], on rightClick [handler], on rightClickOneTime [handler], on doubleRightClick [handler]

on itemSelect

Summary

This handler is executed whenever the user selects a menu item from a menu created using the `create menu` command.

Syntax

```
on itemSelect
  case item1Name
    commands
  case item2Name
    commands
  :
  case itemNName
    commands
end itemSelect
```

Arguments

item1Name…itemNName	The name of the menu item to match, enclosed in quotes.
commands	The list of commands to execute when this menu item is selected.

Description

When the user selects a menu item from a menu created using the `create menu` command, the `on itemSelect` handler is automatically triggered and a check is made to see if the menu item's name matches an existing case statement inside the `on itemSelect` handler. If it does, it executes the commands beneath it. If it does not match, STSTray takes the default action, which is to write the name of the selected menu item out to the output file.

Example

This is an example of a menu defined in the `on openScript` handler triggers selected items in the `on itemSelect` handler:

```
on openScript
  create menu "Greetings,Hello,Remove This Menu Item,-,Exit"
end openScript

on itemSelect
  case "Greetings"
    -- This sends the word "Greetings" to the standard output file
    -- since no commands are defined

  case "Hello"
    write "Hello everyone!" to file "greetings.txt"

  case "Remove This Menu Item"
    -- This redefines the menu with a new call to 'create menu',
    -- effectively removing this menu item
    create menu "Greetings,Hello,-,Exit"
end itemSelect
```

See also

create menu, popup menu

on leftClick (on click)

Summary

This handler is executed whenever the user clicks the icon in the system tray with the left mouse button.

Syntax

```
on leftClick
  commands
end leftClick
```

```
on click
  commands
end click
```

Description

If not defined, the STSTray application will quit when clicked on with the left mouse button under certain circumstances (see Quitting STSTray for more information). You can, however, have STSTray take custom actions based on the click.

Example

This is an example where clicking the icon with the left mouse button will cause the icon to flash:

```
on click
  flash icon "icons/myicon.jpg" every 750 ms
end click
```

See also

on leftClickOneTime [handler], on doubleLeftClick [handler], on doubleLeftClickOneTime [handler], on rightClick [handler], on rightClickOneTime [handler], on doubleRightClick [handler], on doubleRightClickOneTime [handler]

on leftClickOneTime (on clickOneTime)

Summary

This handler is executed the first time a user clicks the icon in the system tray with the left mouse button.

Syntax

```
on leftClickOneTime
  commands
end leftClickOneTime
```

Description

The first time the user clicks the icon in the system tray with the left mouse button, this code will execute. The second and subsequent times the user clicks the icon, the `on leftClick` handler will be executed. Note that you can "reinstall" a one-time handler by adding it again in via another drop script file.

Example

The following example displays a dialog box only the first time the icon is clicked with the left mouse button:

```
on leftClickOneTime
  answer "You clicked me! But you won't see this again..."
end leftClickOneTime
```

See also

on leftClick [handler], on doubleLeftClick [handler], on doubleLeftClickOneTime [handler], on rightClick [handler], on rightClickOneTime [handler], on doubleRightClick [handler], on doubleRightClickOneTime [handler]

on openScript

Summary

This handler is executed whenever STSTray reads a boot script or drop script file that contains this handler.

Syntax

```
on openScript
  commands
end openScript
```

Description

This handler is used to tell STSTray to do something. STSTray reads the boot script file on launch, and then drop scripts subsequently thereafter during its polling mode. This is the main conduit for communicating to STSTray from the outside world.

This handler is generally used on launch of STSTray for setting up menus, defining icons, etc. It is then usually used in subsequent communications for flashing icons, showing balloons, or displaying message dialogs.

Example

This is a basic, simple script:

```
on openScript
  set the icon to "myicons/myapp.ico"
  set the tooltip to "MyApp"
end openScript
```

on rightClick

Summary

This handler is executed whenever the user clicks the icon in the system tray with the right mouse button.

Syntax

```
on rightClick
  commands
end rightClick
```

Description

If a menu has been defined for the icon using `create menu`, the menu will popup automatically when the mouse goes *down* on the icon using the right mouse button. You can define what happens when the right mouse button goes *up* using this handler.

Example

This is an example where clicking the icon with the right mouse button will start an animation of swapping icons:

```
on rightClick
  swap icons "../mail1.ico" and "../mail2.ico"
end rightClick
```

See also

on leftClick [handler], on leftClickOneTime [handler], on doubleLeftClick [handler], on doubleLeftClickOneTime [handler], on rightClickOneTime [handler], on doubleRightClick [handler], on doubleRightClickOneTime [handler]

on rightClickOneTime

Summary

This handler is executed the first time a user clicks the icon in the system tray with the right mouse button.

Syntax

```
on rightClickOneTime
  commands
end rightClickOneTime
```

Description

The first time the user clicks the icon in the system tray with the right mouse button, this code will execute. The second and subsequent times the user clicks the icon, the `on rightClick` handler will be executed. Note that you can "reinstall" a one-time handler by adding it again via another drop script file.

Example

The following example displays a dialog box only the first time the icon is clicked with the right mouse button:

```
on rightClickOneTime
  answer "You clicked me! But you won't see this again..."
end rightClickOneTime
```

See also

on leftClick [handler], on leftClickOneTime [handler], on doubleLeftClick [handler], on doubleLeftClickOneTime [handler], on rightClick [handler], on doubleRightClick [handler], on doubleRightClickOneTime [handler]

on scriptError

Summary

This handler provides a means of trapping script errors and taking action based on this, but only if the `silentErrors` has been set to true (see `set silentErrors`).

Syntax

```
on scriptError
  commands
end scriptError
```

Description

Normally, errors that are generated by STSTray are reported by displaying a dialog box on the screen. This can be a bit embarrassing for end users to see, so there is an option to redirect errors to a log file, or to handle them yourself with this handler.

When STSTray launches, the `silentErrors` are automatically set to false; if you set it to true and an error is generated, STSTray will check to see if there is an `on scriptError` handler that is loaded. If there is, it will trigger that handler and execute whatever commands are inside that handler. If not, errors will either be written to a file called `errors.log`, which is in the same directory as the STSTray executable. If the file already exists it will be appended to; if it does not exist, it will be created.

Example

The following example turns off the display of STSTray's error dialogs, and provides a custom error dialog instead:

```
on openScript
  set the silentErrors to true
end openScript

on scriptError
  answer error "An error was generated performing the last action. Please
   contact MyCompany at 555-1212 and report the error." titled "Error
   Encountered"
end scriptError
```

See also

set silentErrors

on timerEvent

Summary

This handler holds a series of commands that will execute each time the timer that has been set with `set timer` expires.

Syntax

```
on timerEvent
  commands
end timerEvent
```

Description

One common method for using this handler is to execute the `check for updates` command, but it can be used for other things as well (reminders, etc.).

Note that STSTray has only one timer (see comments in the Description section of the `set timer` command).

Example

The following is an example that displays an alert box each time an hour has passed:

```
on openScript
  set the timer to 1 hour
end openScript

on timerEvent
  answer "An hour has passed."
end timerEvent
```

Another example can be seen in the Example section of the `check for updates` command.

See also

check for updates, set timer

Reminder Application

The following example asks you to imagine a fictitious application called “Reminder”, that is a standalone executable with a simple interface that lets you schedule events, and then will alert the user when an event occurs by using the balloon in the system tray (for Windows XP or greater) or a flashing icon followed by an answer box when the icon is clicked (for Windows 95/98/ME/2000).

The idea is that the tray icon will open the user interface of the Reminders application, and allow the user to set up a reminder.

During installation, the STSTray executable is installed in the same directory as the Reminders program, along with an icon for the tray called `Reminder.gif`, and a shortcut to STSTray is installed in the Startup folder. The boot script looks like this:

```
on openScript
  set the icon to "Reminder.gif"
  set the tooltip to "Reminders"
  create menu "Setup Reminders...,-,Exit"
end openScript

on itemSelect
  case "Setup Reminders..."
    launch "Reminder.exe"
end itemSelect
```

The user activates the Reminder application by choosing “Setup Reminders” from the icon in the system tray in order to set up a doctor’s appointment. In the interface, the user sets up a reminder to go off tomorrow at 11:00 am, with a title that says: “Doctor’s appointment at 11:00”, and text that says “Don’t forget to bring your insurance card.” The user is done setting the appointment, and clicks the “Quit” button, and then goes about his/her work.

The Reminders application hides its UI but remains running in the background. Since the program is already running, if the user selected “Setup Reminders” it would launch another instance of the Reminders program, so it needs to *change* the behavior of that menu item. As it goes into the background, it executes this code:

MetaTalk/Transcript

```
on ChangeTrayMenu
  put format("on openScript\ninstall handler:\non itemSelect\ncase "Setup
    Reminders..."\nwrite \"DoSetup\" to file \"Reminder.cmd\"\nend itemSelect\nend
    openScript") into tScript
  put tScript into url("file:newmenu.scp")
end ChangeTrayMenu
```

Visual Basic

```
Private Sub ChangeTrayMenu
  tScript = "on openScript" & vbCrLf & "install hander: " & vbCrLf & "on itemSelect"
    & vbCrLf & "case " & q("Setup Reminders...") & vbCrLf & "write " & q("DoSetup") &
    " to file " & q("Reminder.cmd") & vbCrLf & "end itemSelect" & vbCrLf & "end
    openScript"
  Open (App.Path & "\newmenu.scp") for Output as #1
  Print #1,tScript
  Close #1
end Sub
```

```
Private Function q(ByVal strData as String)
  q = Chr(34) & strData & Chr(34)
End Function
```

This will create a drop script file called `newmenu.scp`, which contains the following contents (white space has been added for clarity):

```
on openScript
  install handler:
    on itemSelect
      write "DoSetup" to file "Reminder.cmd"
    end itemSelect
end openScript
```

When STSTray sees this file, it will open it, and install a replacement `on itemSelect` handler that will write the word “DoSetup” to a file called `Reminder.cmd` instead of launching the Reminders application. To the user, the menu item has not changed.

The Reminders application then goes into its own polling mode, watching for any file by that name, using this polling script:

MetaTalk/Transcript

```
on PollForFile
   if there is a file "Reminder.cmd" then
     put url ("file:Reminder.cmd") into tCommand
     delete file "Reminder.cmd"
     if tCommand is "DoSetup" then
       DisplayInterface  -- handler that will show the UI to the user
     end if
     send PollForFile to me in 100 milliseconds
   end if
end PollForFile
```

Visual Basic

(assumes a Timer object called Timer1 with an Interval of 100.)

```
Private Sub PollForFile
  Timer1.enabled = true
end Sub

Private Sub Timer1_Timer()
  Dim tCommand As String
  tFile = App.Path & "\Reminder.cmd"
  If FileExists(tFile) Then
    Open tFile For Input As #1
    Line Input #1, tCommand
    Close #1
    Kill tFile
    If tCommand = "DoSetup" then
      DisplayInterface  'subroutine to show the UI to the user
    End if
  End If
End Sub

Private Function FileExists(ByVal tPath As String)
  On Error Resume Next
```

```
  temp = Dir(tPath)
  If Err.Number <> 0 Then
    FileExists = False
  End If
  FileExists = Not (temp = "")
End Function
```

When the time comes for the doctor's appointment, the Reminders application calls on the following "generic" script, passing in the title ("Doctor's appointment at 11:00") for the `pTitle` param, and the text ("Don't forget to bring your insurance card.") in the `pText` param:

MetaTalk/Transcript

```
on DoReminder pTitle,pText
   put (the reminderScript of me) into tData
   replace "TITLEPLACEHOLDER" with q(pTitle) in tData
   replace "TEXTPLACEHOLDER" with q(pText) in tData
   put tData into url("file:doReminder.scp")
end DoReminder

function q what
   return quote & what & quote
end q
```

The `reminderScript` custom property contains the following:

```
on openScript
  show info balloon with TEXTPLACEHOLDER titled TITLEPLACEHOLDER
end openScript

on altShowBalloon
  beep
  flash icon
  install handler:
    on clickOneTime
      stop flash
      answer info TEXTPLACEHOLDER with "OK" titled TITLEPLACEHOLDER
    end clickOneTime
end altShowBalloon
```

Visual Basic

(assumes a Text object called Text1.)

```
Private Sub DoReminder(ByVal pTitle as String, ByVal pText as String)
  tData = Text1.text
  tData = Replace(tData,"TITLEPLACEHOLDER",q(pTitle))
  tData = Replace(tData,"TEXTPLACEHOLDER",q(pText))
  Open (App.Path & "\doReminder.scp") for Output as #1
  Print #1,tData
  Close #1
end Sub

Private Function q(ByVal strData as String)
  q = Chr(34) & strData & Chr(34)
End Function
```

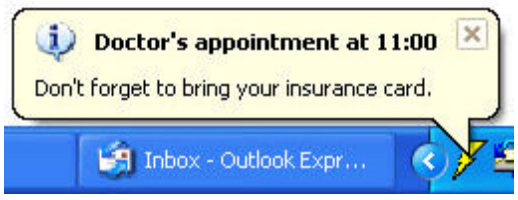
The text field Text1 contains the following:

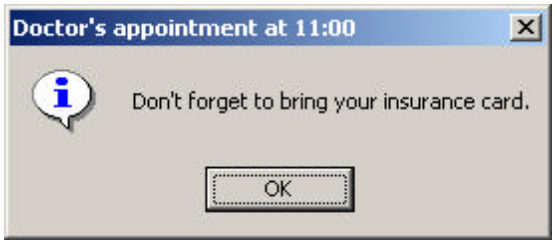
```
on openScript
  show info balloon with TEXTPLACEHOLDER titled TITLEPLACEHOLDER
end openScript

on altShowBalloon
  beep
  flash icon
  install handler:
    on clickOneTime
      stop flash
      answer info TEXTPLACEHOLDER with "OK" titled TITLEPLACEHOLDER
    end clickOneTime
end altShowBalloon
```

So the Reminders application writes out the following TrayScript file to the drop script file `doReminder.scp`:

```
on openScript
  show info balloon with "Don't forget to bring your insurance card." titled
"Doctor's appointment at 11:00"
end openScript

on altShowBalloon
  beep
  flash icon
  install handler:
    on clickOneTime
      stop flash
      answer info "Don't forget to bring your insurance card." with "OK" titled
"Doctor's appointment at 11:00"
    end clickOneTime
end altShowBalloon
```

The reminder is then displayed in one of the following ways (balloon under XP, answer dialog under other systems):





Notifying Users of Product Updates

This example was covered to some degree in the Language Guide above, but is shown here in its completeness for clarity.

The following example assumes you have a product that you sell or distribute, and you wish to be able to notify users whenever an update has been made available via STSTray, and you'd like it to check every 4 hours for an update. For the purposes of this example, the product will be called "Morpheus 1.0".

To prepare for this, you set up a folder on your web site which STSTray will check at specific intervals. This folder is located at `http://www.mycompany.com/morpheus/updates/`, and you leave it empty for right now.

You then prepare your installer, which installs your product and puts the STSTray folder (which contains the STSTray executable) in the same directory as your product.

Your installer also puts a shortcut to the STSTray application in the Startup folder (so it will kick in on each restart of the machine) and your `boot.scp` file looks like this:

```
on openScript
  set the icon to "icons/myproduct.gif"  -- installs custom icon
  create menu "Check for Updates...,Launch Morpheus,-,Exit"
  check "http://www.mycompany.com/morpheus/updates/updates.scp" for updates
  set the timer to 4 hours
end openScript

on itemSelect
  case "Check For Updates..."
    check "http://www.mycompany.com/morpheus/updates/updates.scp" for updates

  case "Launch Morpheus"
    launch "c:\program files\morpheus\morpheus.exe"
end itemSelect

on timerEvent
    check "http://www.mycompany.com/morpheus/updates/updates.scp" for updates
end timerEvent
```

The boot script sets the icon, installs a menu allowing the user to do a manual check for updates, immediately checks for a new update, and then starts a timer that will check for updates every 4 hours.

Since the folder on the web site is currently empty, each time STSTray does a `check for updates`, it doesn't find anything and resets the timer.

Several months later, you have a bug fix update to Morpheus that is version 1.1. You have uploaded the information on the new update to your web site at `http://www.mycompany.com/morpheus/index.htm` and that page also has your downloading instructions, etc. Now all you need to do is notify everyone who has Morpheus 1.0 that there is an update.

You create a notification file called `updates.scp` (the name of the file that the `check for updates` command is looking for) that contains the following TrayScript:

```
on openScript
  set the scriptID to 1000
  show info balloon "A new version of Morpheus (version 1.1) is available. Click
    here to see what's new." titled "Morpheus 1.1 is Available"
end openScript

on balloonClick
  -- Go to the Morpheus web page
  goURL "http://www.mycompany.com/morpheus/index.htm"
end balloonClick
```

```
on altShowBalloon
  flash icon
  install handler:
    on clickOneTime
      stop flashing
      answer info "A new version of Morpheus (version 1.1) is available. Would
        you like to see what's new?" with "YesNo" titled " Morpheus 1.1 is
        Available"
      switch result
        case "Yes"
          goURL "http://www.mycompany.com/morpheus/index.htm"
      end switch
    end clickOneTime
end altShowBalloon
```

Finally, you upload it to `http://www.mycompany.com/morpheus/updates/updates.scp`, the folder that STSTray is checking.

When each customer's copy of STSTray checks for updates, it will see this file there, download it, get the `scriptID` (1000) and check it against what it previously had stored (which was nothing since this is the first update). It is obviously new, so STSTray sets it stored ID to 1000, creating the `ststray.ini` file (since there wasn't one there before) and setting the `LastID` key in the INI file to 1000, and loads the script. It executes the `show balloon` command, which will display the balloon on Windows XP or greater – if they click on the balloon, it will take them to the web page with more information about the product update. If they don't have Windows XP or greater, the `on altShowBalloon` handler will kick in and it will flash the icon in the tray, and install a one-time handler that will activate when the user clicks the icon with their mouse. When they click, it will execute the commands in the installed `on clickOneTime` handler, and will bring up an `answer` dialog box, letting them know a new version of Morpheus is available. If they click "Yes", they will be taken to the web page with more information on the product update. If they click "No", the dialog box goes away. In either case, STSTray will go back to checking for updates every 4 hours.

As you can see, this is a really easy way to notify users of product updates, or anything else you'd like to bring to their attention. And if, for some reason, you need to modify the tray menu permanently, or change what happens when STSTray is launched, you can download a file which deletes the current `boot.scp` with the `delete` command, and then creates a new one with the `write` command.