



TaskGhost Users Guide

Version 1.1

Monday, April 16, 2007

**This software is provided as-is.
There are no warranties, expressed or implied.**

**Copyright© 2006 Tomasello Software, LLC.
All rights reserved**

**TaskGhost® is a Registered Trademark of Tomasello Software
LLC.**

Table of Contents

Table of Contents.....	1
Auto-Execution Scripts.....	1
Command-Line Syntax	2
Fixing “File Not Found” Errors.....	2
Interactive Control Panel.....	5
Email.....	6
Extended Cron Syntax	6
Run as a System Service.....	8
Script Syntax.....	11
Security	12
System Tray Icon	13

Auto-Execution Scripts

When TaskGhost starts it will perform the following steps in order:

1. Load and run AutoRun.vbs if it exists. If it does not exist, TaskGhost will simply enter a Ready state.

2. AutoRun.vbs will run Schedule.vbs which is the main job dispatcher. When and if Schedule.vbs exits, TaskGhost will return to a Ready State.
3. After AutoRun.vbs is run, but possibly before it completes, TaskGhost will load and run any script specified on the command-line.

If no scripts are found, TaskGhost will simply go into 'Yellow Icon' mode meaning "there is no work to be done."

Suggestion: Because debugging scripts while running TaskGhost as a System Service is somewhat complicated, I recommend testing your AutoRun.vbs modifications with TaskGhost running in application mode.

For example:

TaskGhost -application

Command-Line Syntax

TaskGhost [-install [-account account_name -password password]] [-uninstall] [-help] [-version] [-application]

Note: Most of the command line arguments are also available as -action directives.

Please see the expanded usage information in the [-actions section](#).



-install, [-account account_name -password password]

This switch causes TaskGhost to register itself as a Windows System Service.



-uninstall, This switch causes TaskGhost to unregister itself as a Windows System Service.



-help, get a brief summary of the TaskGhost command line switches.



-version, Get the version information and install state of TaskGhost



-application, force TaskGhost to start-up in regular application mode after being registered as a system service with the -install switch.

Fixing "File Not Found" Errors

How TaskGhost finds its scripts and executable programs

Most computer systems today have a PATH environment variable, to store information about the location of programs.

The environment variable is nothing more than a list of directories to check when looking for a program to execute. For example:

C:\BIN;C:\WINDOWS;C:\WINDOWS\SYSTEM32;

TaskGhost uses the low-level Win32 facilities for Process Creation, and these facilities rely on the PATH environment variable.

When a program starts up, it gets its own private copy of the environment for the current account. So for instance, when someone logs into their computer and launches Microsoft Word, Word gets a copy of the environment for the current account.

This same rule applies to all other programs as well. When TaskGhost is launched, it too gets a copy of the local environment.

Upon startup, TaskGhost looks for the directory in which TaskGhost.exe resides. We'll call this directory [ROOT]. TaskGhost then adds the following path elements to its own private copy of the environment:

[ROOT]\Scripts
[ROOT]\Sample Scripts
[ROOT]\Diag

So the above path might now look like:

**C:\PROGRAM FILES\TOMASELLO
SOFTWARE\TASKGHOST\SCRIPTS;C:\PROGRAM FILES\TOMASELLO
SOFTWARE\TASKGHOST\SAMPLE SCRIPTS;C:\PROGRAM
FILES\TOMASELLO SOFTWARE\TASKGHOST\DIAG;
C:\BIN;C:\WINDOWS;C:\WINDOWS\SYSTEM32;**

TaskGhost patches the PATH environment variable on startup so as to guarantee that files in the Scripts, Sample Scripts, and Diag directories will always be found.

Finding files on YOUR Computer

Even though TaskGhost knows where all of its files are, it may not know where all of your files are. Probably the number one complaint people have when running a scheduling program are the "File not found" errors. This is primarily due to the following: The PATH variable (for the current account) does not contain the directory in which the desired file resides.

This seems simple enough to fix, right?

Well the complication comes in when you are running TaskGhost as a system service, or running TaskGhost logged into a specific user account, or having TaskGhost launch programs *in the security context of another user*.

The important thing to remember is that the PATH environment variable for USER-A is not necessarily the same as the PATH environment variable for USER-B. Also, the PATH environment variable for the LocalSystem account

(which is the account TaskGhost runs in when TaskGhost is running as a system service) will likely not be even close to USER-A's or USER-B's PATH environment variables.

The reason for this is that when you install a program, the installation software sets the environment variables for the *Current User* only and not every user and account on the computer; and this is as it should be. You as a system administrator just need to be aware of this fact.

To successfully schedule programs to run under all of the above scenarios, you can adopt one of the following approaches.

1. Insure all accounts have all the environment settings of everyone else.
A terrible solution and only listed here to dissuade you from considering it.
2. Place the path information to all the programs you plan to run in the LocalSystem's environment (PATH variable.)
This is actually a reasonable way to go. When you set the System environment variables, they show-through all accounts, so it wouldn't matter if TaskGhost was running as USER-A, or USER-B, all the program paths would be available.
3. Patch the TaskGhost PATH environment variable at startup to contain all the path information you want.
This is probably the best solution because it localizes the solution to TaskGhost instead of the machine.
See the functions: [GetEnvironmentVariable](#), [SetEnvironmentVariable](#)
4. Always use explicit paths when launching a program.
This is the low-impact solution, but adds a little 'hard-codedness' to your scripts.

Interactive Control Panel



The TaskGhost UI has an output area where it writes various status as well as print output from the scripts.

Under the output area, there is a group of buttons:

(Suspend) (Hide) (Help)

(Stop) (Load) (About)



Pressing the Suspend / Resume button will toggle the state of the running jobs between suspended and resumed.

Suspended jobs are not truly suspended, but instead looping idly waiting for a resume message.

If you're interested, you can see this logic in Schedule.vbs.

Pressing the Stop button will stop and clear all jobs.



When the jobs are stopped they can not be resumed. To restart stopped jobs, you must reload and run the script(s).



Pressing the Load button will load and run a new script from disk while leaving the current jobs running. Loading a script effectively appends new jobs to a currently running jobs list.



Pressing the Hide button will hide the TaskGhost User Interface. The TaskGhost User Interface will not reappear unless all jobs expire or there is an error.

When TaskGhost is running as a system service, the hide button is how the user interface is dismissed.



Pressing the Help button will provide you with several syntax examples to help you get going fast.

Pressing the About button will provide the usual version information as well as a Legal notice and a screen giving the user an opportunity to register their copy of TaskGhost.

Email

Overview

EMail alerts can be a very powerful tool to use when building your automation scripts. This is because alerts can be emailed at any time of the day or night to alert system administrators of system failures or other situations that need immediate attention.

Alerts can also be used to notify an administrator that a backup has finished, a system has shutdown, etc.

With the new generation of PDAs and cell phones that can be used to check e-mail, the ability to programmatically send alerts using this technology becomes even more valuable.

How it's done

TaskGhost ships with a sample script for sending email, here it is:

```
Option Explicit ' Force declarations for variables
Sub main(args)
Dim objEmail
Set objEmail = CreateObject("CDO.Message")
objEmail.From = "sys_monitor@my_corp.com"
objEmail.To = "support@my_corp.com"
objEmail.Subject = "server NORBERT offline"
objEmail.Textbody = "server NORBERT is not responding."
objEmail.Send
TGCtrl.Print "Mail sent."
End Sub
```

For this script to work, the SMTP service must be installed on the computer where the script is being run.

That's really there is to it.

You can use this technique for sending automated emails and alerts.

See Also:

[SendMail](#),

EEmail.vbs in the TaskGhost\Sample Scripts directory.

Extended Cron Syntax

TaskGhost contains a very powerful pattern matching engine that is used for matching *the current time*.

TaskGhost supports an extension of the UNIX CRON standard.

Standard Cron Syntax

TaskGhost's UNIX cron-style matcher is based on Paul Vixie's work which is upward compatible with the V7 standard. There is however a difference; The

various versions of UNIX can't decide if Sunday is 0 or 7 when specifying days of the week. For TaskGhost Days Of The Week, values range between 1 and 7, where 1=Sunday, 2=Monday, and so on.

The TaskGhost pattern matching engine recognizes the following fields:

field	meaning	allowed values
1	minute	0-59
2	hour	0-23
3	day of month	1-31
4	month	1-12
5	day of week	1-7 (1=Sun, etc.)

A field may be an asterisk '*', which always stands for "first-last".

Ranges of numbers are allowed. Ranges are two numbers separated with a hyphen. The specified range is inclusive. For example, 8-11 for an "hours" entry specifies execution at hours 8, 9, 10 and 11.

Lists are allowed. A list is a set of numbers (or ranges) separated by commas. Examples: "1,2,5,9", "0-4,8-12".

Extended Cron Syntax

In addition to the standard CRON fields described above, The TaskGhost pattern matching engine also recognizes the following optional fields:

field	meaning	allowed values
6	year	100-9999
7	day of year	1-366, (1=Jan 1, etc.)
8	second	0-59

When an optional field is omitted, it is treated as *ignore* by the expression evaluator as opposed to the '*' which stands for *match all*.

Like the asterisk '*', the under bar '_' also has special meaning and is used to *ignore* imbedded fields.

For instance:



"* * * * *" : Fields 6, 7, and 8 are implicitly set to ignore.



"* * * * * _ _ _ " : Fields 6, 7, and 8 are explicitly set to ignore.



"* * * * * _ * *" : embedded Field 6, is explicitly set to ignore.

The '_' specifier is not terribly useful for scheduling standard TaskGhost jobs, but it becomes necessary when scheduling Windows AT commands. This is because the internal structures for Windows AT commands do not support the richness of expression that CRON syntax does.

Cron Syntax for scheduling AT commands

The native Windows Scheduler, sometimes called "AT commands" is the scheduling facility that ships with Windows. You can schedule these jobs with either AT.EXE at a DOS prompt, or the Schedule application, both of which ship with Windows.

TaskGhost gives you a better way to create, list, and delete these native Windows jobs.

Because the Windows AT scheduler only understands "days of the week", "days of the month" and "time", a few rules should be noted when creating Windows AT command schedules.



Field 1&2 (hour, minute) must be numeric values - no ranges, lists, or '*' allowed here.



Field 3 (day of the month) usual CRON syntax.



Field 4 (month) must explicitly set to ignore with the '_' character.



Field 5 (day of the week) usual CRON syntax.



Fields 6&7 (year, day of the year) must either be omitted or explicitly set to ignore with the '_' character.



Field 8 (second) must be a numeric value - no ranges or '*' allowed here. Lists are ok. (if supplied.)

Note: At the time of this writing, The Windows Scheduler has a bug whereby the 'seconds' field is ignored.

You may read more about this particular Windows bug here: [Microsoft Knowledge Base Article - 276381 \(KB276381\)](#)

Please see the tutorial section on [Advanced Scheduling using patterns](#).

Run as a System Service

Overview





A System Service is a program that is given permission to run at Operating System load time, and with credentials different than that of

the logged in user. This special permission allows a system service to remain resident in memory even while users login and logout.





System services can run in either the special "LocalSystem" account, or logged in as a specific user.

Here is a comparison of the two types of system services (and the two ways TaskGhost can be configured to run.)

LocalSystem

-  PRO: Can interact with the user by displaying a standard Windows Graphical User Interface.
-  PRO: Easy to setup. Easy to diagnose problems (because you can see the interface.)
-  CON: Certain standard applications have problems when run from a LocalSystem account: Browsers, email clients, etc..
-  CON: The environment: network drives, PATH, etc., is usually much different from the user's account. (Adds a complication to diagnosing script problems.)

User Account

-  CON: Cannot interact with the user by displaying a standard Windows Graphical User Interface.
-  CON: Not so easy to diagnose problems (because you cannot see the interface.)
-  PRO: Most standard applications have no problems when run from a User Account.
-  PRO: The environment: network drives, PATH, etc., is the same as user's account. (Makes it easy to diagnose script problems.)

You probably noticed that the strengths of one of the approaches is the weakness of the other, and visa versa.

Here are some considerations when choosing which approach to use: If you answer YES to any of the following, then you should configure TaskGhost to run as a system service *logged in as a specific user*.

1. Will you want to send email from the script?
2. Will you be accessing network drives?
3. Will you be launching a browser from the script?
4. Will the running script be heavily dependant on the user's environment (mapped drives, PATH, etc..)

If you answered NO to the above, then you can probably run TaskGhost in the LocalSystem account. This is the preferred mechanism as you get the

interactive console. Otherwise you will need to run TaskGhost logged into a user account.

Note: The close box is disabled when TaskGhost is being runs as a system service, because the correct mechanism for shutting down such a service is through the Windows Control Panel / Services applet.

Getting your System Service up-and-running

1. Install the product as usual.
2. If you wish to install TaskGhost as a system service (LocalSystem)
Run TaskGhost with the -install.
Example: "TaskGhost -install"
3. If you wish to install TaskGhost as a system service (Domain User)
Run TaskGhost with the -install -account xxx -password yyy switches.
Example: "TaskGhost -install -account luket -password rocketdog"
4. Alternatively, you can install TaskGhost as a system service running in the LocalSystem and go to the Windows Control Panel / Services applet and specify a user name and password.

Assuming you got the script right, you should be ready to go! You can now either reboot to get TaskGhost to load or use the Control Panel/Services applet to start TaskGhost.

Troubleshooting your new System Service

The most common problems with running a System Service have to do with the PATH environment variable. It is important to remember that the PATH information in your USER account is not the same PATH information in the "LocalSystem" account, which is the account TaskGhost logs into when it is run as a service.

Another common problem is setting the system PATH correctly, but forgetting to reboot the computer so that the new PATH information can be used by the System Service.

And finally, TaskGhost looks for the [AutoRun.vbs](#) in the TaskGhost\Scripts directory. If the [AutoRun.vbs](#) cannot be found, TaskGhost can not start any jobs and will be left running in an idle state.

For more help troubleshooting TaskGhost when installed as a System Service, please check the TaskGhost FAQ.

Script Syntax

The Basic Job

The basic TaskGhost job has two parts:

The Trigger

The trigger is a one line entry usually made in the [Schedule.vbs](#) file and looks something like this:

```
" Run "defrag.exe" on Mon, Wed, Fri at 1930 (7:30PM).  
If TGCtrl.CheckTime (8, tc, "30 19 * * 2,4,6") = true Then TGCtrl.Run 0, "CreateProcess.vbs",  
"c:\bin\defrag.exe"
```

The above trigger says: If the current time is "**30 19 * * 2,4,6**", which in English means; Mon, Wed, Fri at 1930 (7:30PM), then run the program [c:\bin\defrag.exe](#).

The CheckTime function takes a [CRON specification](#) and checks it against the current time. If the current time matches the CRON specification, then the script (or command) after the 'Then' is run.

In this example we run the script called [CreateProcess.vbs](#) to create a new process (run a program) and handle any failures. We could certainly have launched the program from Schedule.vbs directly, but then we would need error handling code and print statements making the main scheduling table somewhat hard to read.

This two step process the fundamental TaskGhost job construction philosophy;

5. **Create a table entry in Schdule.vbs**
6. **Call a script to carry out the job**

One possible exception to this rule might be to use a table entry to perform a simple print statement as shown in the following example:

```
" Print out a message at 9AM Christmas morning.  
If TGCtrl.CheckTime (0, tc, "0 9 25 12 *") = true Then TGCtrl.Print ("It's Christmas!")
```

The Job

Jobs are just VBScripts that you launch in response to a [trigger](#).

The job used in the example above is a reusable script for launching external programs like .EXE.

Have a look at what [CreateProcess.vbs](#) looks like.

We call this script reusable because it doesn't really matter what program (.EXE) it's running. It simply launches whatever program is passed to it on the commandline.

Q. If you're at all familiar with VB or VBScript, you're probably wondering what [Function](#) or [Sub](#) are we running because all that is being specified is the script name?

A. In order to give TaskGhost a uniform and consistent model for it's internal schedule management, TaskGhost has adopted an approach not unlike C/C++ where a 'program' has a fixed entry point. For TaskGhost this is the *main* [Function](#).

So all *.VBS scripts should be viewed as a complete program, containing all of the necessary procedures and data to accomplish a task. When TaskGhost loads one of these .VBS files, it immediately looks for the *main Function* and calls it.

Please see the TaskGhost\Scripts directory for more job samples.

Security

TaskGhost provides security for confidential information, such as passwords, by using a data encryption algorithm based on the RC2 64-bit symmetric block cipher.

Data encryption is performed when [storing environment variables](#) when the special encryption flag is used.

The data is then decrypted when the same [variable is retrieved](#).

TaskGhost environment variables can be either transient, or persistent.

Handling Passwords

A typical way a user might manage her secret data would be to write a small script to store persistent variables with information like user name, password, server, etc. Then, after the data is saved, simply delete or otherwise hide the script that stored these values.

This sample script shows the storing and retrieving of encrypted variables. You'll notice that aside from the specification of the encryption flag, encryption usage is transparent.

[Option Explicit](#)

' Force declarations for variables

[Sub](#) main(args)

```
Dim user
```

```
Dim domain
```

```
Dim password
```

```
.....
```

```
" lets secure these values with encryption
```

```
"          (only available in the registered version of TaskGhost.)
```

```
" NOTE: If you wish to use encryption to protect things such as
```

```
"          passwords etc., write a script to first enter the values into
```

```
"          the persistent store, then delete or otherwise hide that script.
```

```
" You will be able to then retrieve and decrypt the values at a
```

```
"          future date.
```

```
Dim Flags
```

```
Flags = TSE_ENV_PERSISTENT
```

```
Flags = Flags Or TSE_ENV_CRYPTO
```

```
" first write the encrypted values to the persistent store
```

```
TGCtrl.SetEnvironmentVariable Flags, "MY_USER", "luket"
```

```
TGCtrl.SetEnvironmentVariable Flags, "MY_DOMAIN", "."
TGCtrl.SetEnvironmentVariable Flags, "MY_PASSWORD", "rocketdog"

" now retrieve and decrypt the values from the persistent store
user = TGCtrl.GetEnvironmentVariable (Flags, "MY_USER")
domain = TGCtrl.GetEnvironmentVariable (Flags, "MY_DOMAIN")
password = TGCtrl.GetEnvironmentVariable (Flags, "MY_PASSWORD")

" lets double check the results
TGCtrl.Print"MY_USER = " + user
TGCtrl.Print"MY_DOMAIN = " + domain
TGCtrl.Print"MY_PASSWORD = " + password
```

End Sub

Note: Data encryption is only available in the Registered version of TaskGhost.

System Tray Icon

TaskGhost places an icon in the system tray of the taskbar.

This icon has several uses. First of all it gives you visual feedback as to the running state of TaskGhost. It also allows you to display the TaskGhost window by double clicking the icon or by right-clicking the icon and selecting 'Show' from the menu.

Yellow Icon: TaskGhost is in starting-up mode. The Yellow icon will also persist if TaskGhost was not given any tasks to perform. So if you are running as a service and notice a persistent yellow icon, then there is a good chance TaskGhost could not find the [AutoRun.vbs](#) file.

Green Icon: TaskGhost is currently running a job. If TaskGhost *should* be running and you see a green icon, then you can rest easy.

Red Icon: When TaskGhost displays a red Icon, it simply means that TaskGhost has finished all scheduled tasks and is once again in a quiescent state.