Programmatic Support for Windows NT Services

Note that wherever this document refers to Windows NT, the comments apply to both Windows NT 4, Windows 2000, and later versions, unless stated otherwise.
You can find the latest version of this document on-line at .
You can find a Services Quick Start Guide at .

Windows NT and Windows 2000 provides support for special programs called services. NT services run in the background, usually without direct user interaction, often started automatically at some point in the boot process. They typically run under the special LocalSystem account which means that they do not need to be associated with a username and password. (This also places some security restrictions on what services can do by default.)

Services are analogous to Unix "daemons," but formalised into the operating system. This formal role of NT services brings some advantages to the system administrator. For example, they can be configured to start automatically at various points in the boot process, or can be controlled - locally or remotely - using `NET START`, Control Panel Services or programs such as ServPanel.

The downside of this is that ordinary programs will not run as NT services. A service program needs to establish and maintain contact with the NT Service Control Manager) (SCM. This requires special coding endemic to the program: if a service does not maintain contact with the SCM, NT assumes it has failed and shuts it down.

As an alternative to writing special code, the Windows NT Resource Kit provides a utility called `SRVANY.EXE` ' . `SRVANY.EXE` will start an ordinary program in the context of an NT service. However it is pretty limited and does not address some of the constraints of NT Services:

- NT Services run using the default (system) environment and directory. However often it is necessary to run such programs with their own environment values.
- There is no facility to automatically restart services which crash (as per Unix `init`).
- `SRVANY.EXE` is crude in its handling of service startup and shutdown. It assumes that the program is running as soon as it starts, and has no way to shut down the program other than by terminating it.

Because of these limitations I wrote two utilities, SVC and SRVSTART. **SVC** allows you to install, modify or remove Windows NT services. **SRVSTART** allows you to run executable programs as if they were services.

Between them they give you far greater management and flexibility than is provided by the Windows NT and the NT Resource Kit.

1 Copyright and Distribution

2 Effectivity

This article applies to the following versions of software:

Software	Version	Platform
SVC	1.0	Windows NT 4 (Service Pack 4+) Windows 2000 (1st Release)
SRVSTART	1.1	Windows NT 4 (Service Pack 4+) Windows 2000 (1st Release)

Table 1: Effectivity

You can check the version of the **SRVSTART** DLL or executable by running the following shell command against it (new in version 1.1):

```
find "***" srvstart.dll
find "***" srvstart.exe
```

You will see output similar to the following:

```
*** SRVSTART version 1.10 Copyright (C) 1998 - 2000 Nick Rozanski (##############)
```

SRVSTART is a rewrite (in C++) of an earlier version of the program called `SYBSTART`. The new version contains a number of enhancements and extra features.

3 SVC.EXE - Windows NT Service Management

SVC is a simple command-line utility which allows you to install, modify or remove Windows NT Services

3.1 Pre-Requisites

SVC runs on Windows NT (not 95 or 98). To run it you require the following files somewhere in your PATH.

- `svc.exe` - **SVC** executable
- `msvcrt.dll` - Microsoft C++ run-time library

Both of these components are available from my website.

3.2 Synopsis

To run **SVC**, type `svc.exe` at a console prompt.

3.3 Operation

SVC has the following actions:

- `l` - list existing services
- `d` - display details of a existing service
- `i` - install a new service
- `m` - modify an existing service
- `r` - remove an existing service.

SVC will prompt you for all responses. At any point you can type `?` (question mark) for a short explanatory help message.

If you are having trouble running **SVC**, you can try running `svc.exe -d`. This prints out internal **SVC** debug messages as it runs.

If you don't like **SVC**'s command-line interface, I recommend ServPanel from Ballard Software. This is an easy-to-use GUI utility which allows you to install and modify services, and the latest version has pretty much all the install functionality of **SVC**. It also has a very nice interface for starting and stopping services individually or in groups.

3.4 Restrictions

Because modifying services is a potentially destructive activity, **SVC** is deliberately restricted in what it can do. A service to be installed, modified or removed must have the following characteristics:

1. It must be a service on the local machine.
2. It must be of type `WIN32_OWN_PROCESS` (ie not shared).
3. It must be of start type 'automatic,' 'demand' or 'disabled'.

These restrictions are present for your own safety!

3.5 Warning

Modify Windows NT service settings at your own peril.

Making incorrect changes can render Windows NT unuseable.

If you don't know what you are doing, don't mess about with services!

3.6 Version History

Version 1.0 (30 June 1998)

First release version.

4 SRVSTART.EXE - Windows NT Service Execution

SRVSTART is a Win32 executable and DLL which allows you to run commands as if they were Windows NT services. It also has some features to enhance the running of ordinary console commands (prompting for parameters etc).

You can also use **SRVSTART** to install or remove a service which is based on the `SRVSTART.EXE` executable (new in version 1.1).

You can find a Services Quick Start Guide at .

4.1 Pre-Requisites

SRVSTART runs on Windows NT, 2000, 2003 and XP. To run it you require the following files somewhere in your PATH.

- `srvstart.exe` - **SRVSTART** executable
- `srvstart.dll` - **SRVSTART** library
- `logger.dll` - logger library
- `msvcrt.dll` - Microsoft C++ run-time library

All of these components are available from my website.

4.2 Synopsis

At run time, **SRVSTART** operates in one of two modes.

- It can be used to run an ordinary command (executable program batch file). In this command mode, **SRVSTART** can prompt the user for the values of command-line parameters such as passwords.
- It can be used to run an executable program in the context of a Windows NT service (service mode. **SRVSTART** will itself handle all of the interactions with the NT Service Control Manager (SCM). It is not necessary for the program to include any service management code.

Support for installing and removing services is described later.

4.2.1 Command Mode

Use the following syntax to run a program in command mode.

```
srvstart cmd window_title [ options ... ] program [ program_parameters ... ]
```

- The `cmd` keyword tells **SRVSTART** that this is command mode.
- *`window_title`* will be displayed (in most cases) in the window's title bar when the command runs. Enclose it in quotes if it contains spaces.
- *`program`* and *`program_parameters ...`* define the command to run.
- The *`options`* are defined below.

4.2.2 Service Mode

Use the following syntax to run a program in service mode.

```
srvstart [ svc ] service_name [ options ... ] program [ program_parameters ... ]
```

- The `svc` keyword tells **SRVSTART** that this is service mode. This keyword is optional (new in version 1.1).
- *`service_name`*) is the short (internal) name of the service. (You will have defined this when you installed the service using **SVC**.)
- *`program`* and *`program_parameters ...`* define the command to run.
- The *`options`* are defined below.

4.2.2.1 Note

The `srvstart` syntax is never entered at the command line. Rather, use this syntax as the command line when installing a service (eg in response to **SVC**'s `Enter binary path name` prompt).

For testing, you can supply the keyword `any` instead of `cmd` or `svc`. This will attempt to run the program as a service (keyword `svc`) and if this fails run it as a console command (keyword `cmd`).

4.3 Operation

SRVSTART does the following.

1. If it is running in service mode, it connects to the SCM.
2. It sets `%PATH%`, `%LIB%` and any other environment variables which have been defined in its command line or control file.
3. If it is running in command mode, it optionally prompts the user for input in response to supplied prompts.
4. It starts the command which has been supplied to it.
5. If it is running in service mode, it:
    - optionally waits to let the started process initialise
    - notifies the SCM that the service has started
    - waits for requests from the SCM (eg to interrogate or stop the service) and acts on these
    - if a "shutdown" request is received, it shuts down the program, and exits
    - periodically checks the started command to see if it is still running; if it has finished, it notifies the SCM that it has stopped, and exits.

4.4 Options

The following **SRVSTART** options apply to command and service modes.

-c *controlfile*

retrieve **SRVSTART** options from the file *`controlfile`*

-d *level*

set the **SRVSTART** debug level: `0`=none, `1`=normal, `2`=verbose

-e *var=value*

set the environment variable *`var`* equal to *`value`* before running the program

-h

display help message to `stdout` and exit

-l *libdir*

set the value of the `%LIB%` environment variable to *`libdir`*

-o *target*

define where **SRVSTART** will write debug messages (`-`, `LOG` or *`pathname`*)

-p *path*

set the value of the `%PATH%` environment variable to *`path`*

-q *sybase*

assign a default `%PATH%` based on this value of *`sybase`* instead of value supplied using `-s`

-s *sybase*

set the value of the `%SYBASE%` environment variable to *`sybase`* (equivalent to `-eSYBASE=`*`sybase`*)

-x *priority*

start the command at the given execution priority(`idle`, `normal`, `high`, `real`)

The following **SRVSTART** options apply to command mode only.

-m

start command in minimised new window

-w

start command in new window

The following **SRVSTART** options apply to service mode only.

-t *seconds*

program status check interval in seconds

-y *seconds*

how long **SRVSTART** waits before reportinga "started" status to the NT Service Control Manager

4.4.1 Environment Options

Options `-s`, `-p`, `-q`, `-l` and `-e` control the environment in which the program runs.

If no `-p` option is given, a default path of the form:

```
%SYBASE%\install;SYBASE%\bin;%SYBASE%\dll;%SystemRoot%;%SystemRoot%\system32
```

is used. (`%SystemRoot%` is the Windows NT root directory, eg `C:\WINNT`).

The `-q` option assigns a path as above, but using the value supplied to `-q`, instead of `%SYBASE%`. For example, `-q C:\NEWSYB` would assign a path of the form `C:\NEWSYB\install;C:\NEWSYB\bin;...` etc.

Environment values which contain embedded environment variables will have these substituted, but only if they are already defined. For example:

```
... -e myvar1=myvalue1 -e myvar2=my_%myvar1%_val ...
```

will set `myvar1` to `myvalue1` and `myvar2` to `my_myvalue1_val`. However

```
... -e myvar2=my_%myvar1%_val -e myvar1=myvalue1 ...
```

will set `myvar2` to `my_%myvar1%_val` (since `myvar1` is not defined at this point.

Note that in service mode, the only environment variables available when **SRVSTART** starts are the system environment variables. (These are the environment variables in the *upper* list box in In Control Panel|System|Environment.) If the service is started using a named Windows NT account, then the environment for that account will also available.

4.4.2 Debugging Options

Options `-d`, `-o`, and `-h` control **SRVSTART** debugging.

- `-d 0` prevents any output other than error messages.
- `-d 1` outputs a few informational messages.
- `-d 2` outputs a large number of debug messages.

Note that you must use the Debug executables for level 2 (you can find these in the `Debug` directory of the full distribution).

`-o` *`target`* causes debug messages to be sent to `target`.

- If `target` is `-` (hyphen) messages will go to `stdout`. This does not apply to service mode (messages will just disappear).
- If `target` is `LOG` (uppercase) debug messages will be sent to the Windows NT Event Log.
- Otherwise, `target` is assumed to be a pathname. Debug messages will be appended to this file.

By default, **SRVSTART** logs error messages to the Event Log in service mode, and to `stdout` otherwise.

`-h` displays help information and exits. (The other **SRVSTART** parameters need not be supplied in this case.)

4.4.3 Startup Options

`-x` specifies the priority at which the program should run.

- `idle` to run only when the CPU is otherwise idle
- `normal` to run at normal priority
- `high` to run at high priority
- `real` to run at real time priority

It is ignored in command mode unless `-w` is supplied.

In command mode, `-w` opens a new console window to run the program, and `-m` opens this window minimised. These options are ignored in service mode.

4.4.4 Service Management Options

If the `-y` *`seconds`* option is specified, **SRVSTART** waits this number of seconds before reporting to the Windows NT Service Control Manager that the service has started. Use this option if the command takes a long time to initialise (default zero).

In service mode, **SRVSTART** regularly checks the process it has started to see if it is still running. If it has finished, then **SRVSTART** reports a "service stopped" status to the Service Control Manager and then exits. The `-t` *`seconds`* option defines how often this check is done (default every second).

4.5 Control File

`-c` *`controlfile`* specifies that **SRVSTART** should get its options from *`controlfile`*.

This is a text file with one option per line. Options are grouped in sections, with the section name (which is the window or service name) in square brackets. Options before any section apply to all commands.

All section lines are of the form `[`*`service_or_window_name`*`]`. All option lines are of the form *`keyword=value`*. Blank lines, and comments lines (starting *#*) are ignored.

For example:

```
# comment
keyword=value
...

[service_or_window_name_1]
keyword=value
keyword=value
...

[service_or_window_name_2]
keyword=value
...
```

SRVSTART reads the control file, applying all keywords before the first section. It then finds the section whose name matches the supplied window or service name, and applies all keywords in that section.

When a control file is used, the *`program`* and *`program_parameters`* can be omitted from the command line.

Note that **SRVSTART** applies environment variable substitution to all keyword values which are filenames, pathnames or directories (new in version 1.1). For this to work, the environment variable must already be defined (either globally to Windows NT or using the `env` directive) at the time that the directive is read.

For example, the directive `debug_out=%TEMP%\myservice.out` will log debug output to the file `myservice.out` in the Windows temporary directory (usually `C:\TEMP`).

You *must* provide a `startup` directive for each service which starts using a control file. All other directives are optional.

4.5.1 Control File Keywords (Command-Line Equivalents)

The following keywords replace the **SRVSTART** command-line options.

debug=*level*

same as `-d` *level*

debug_out=*target*

same as `-o` *target*; additionally, if `target` is a path name whose first character is > (greater than), **SRVSTART** will truncate the file before writing to it (new in version 1.1)

env=*var=value*

same as `-e` *var=value*

lib=*libdir*

same as `-l` *libdir*

minimised={y|n}

same as `-m`

new_window={y|n}

same as `-w`

path=*path*

same as `-p` *path*

priority=*priority*

same as `-x` *priority*

sybase=*sybase*

same as `-s` *sybase*

sybpath=*path*

same as `-q` *path*

startup_delay=*seconds*

same as `-y` *seconds*

wait_time=*seconds*

same as `-t` *seconds*

4.5.2 Control File Keywords (Startup and Shutdown)

The following keywords are used to define **SRVSTART** commands. Apart from `startup` they have no command-line equivalents.

startup=*program* [*program_parameters* ...]

This defines the service program command. It replaces the *program* and *program_parameters* which are supplied on the **SRVSTART** command line.

startup_dir=*path*

This defines the startup directory. It should be a full pathname including a drive letter.

wait=*program* [*program_parameters* ...]

This defines a command that **SRVSTART** will run after starting the service program (service mode only).

This should wait for the service program to enter a "running" state (eg wait for a database server to complete recovery). It should exit with a status of `0` once the service program is up and running. It should exit with a non-zero status if the service program has failed or is never going to enter a running status. Once this command has exited with a status of `0`, **SRVSTART** considers that the service program is running.

shutdown_method={kill | command | winmessage}

This defines the action that **SRVSTART** will take to shutdown the service program (service mode only). **SRVSTART** will take this action if it receives a "shutdown" request from the SCM (eg a user runs `NET STOP` or stops the service using Control Panel | Services).

- For `shutdown_method=kill`, **SRVSTART** will shut down the service program using the Win32 `TerminateProcess()` API.
- For `shutdown_method=command`, **SRVSTART** will run the command given by the `shutdown directive`.
- For `shutdown_method=winmessage`, **SRVSTART** will send a Windows `CLOSE` message to all Windows opened by the service program (new in version 1.1).

`kill` is the default, and this directive can be omitted. Note that `kill` is equivalent to a Unix `kill -9` and leaves DLLs in an undefined state (ie it does not call the DLL termination routines). I have not to date observed any problems with this (but it clearly depends on the program you are shutting down).

Note also that `winmessage` will not work for Win32 console programs.

shutdown=*program* [*program_parameters ...*]

This defines a command to shut down the service program (service mode only). **SRVSTART** will run this command if it receives a "shutdown" request from the SCM (eg a user runs `NET STOP`) or stops the service using Control Panel | Services.

If this directive is provided, then `shutdown_method=command` is implied and can be omitted.

auto_restart={y|n}
restart_interval=*seconds*

If `auto_restart` is set, then **SRVSTART** will restart the service program if it exits for any reason. (The assumption here is that the service has crashed.) If `restart_interval` is defined, then before restarting, **SRVSTART** will wait *`seconds`* seconds.

`auto_restart` does not, of course, restart the service program if it is stopped by request (eg `NET STOP` or Control Panel|Services).

`auto_restart` does not restart the service program if it thinks that Windows NT is shutting down (new in version 1.1). Unfortunately it does not appear to be possible to determine this unambiguously. If you are irritated by services restarting during Windows NT shutdown, then increase the value of `restart_interval` to, say, a minute.

4.5.3 Control File Keywords (Drive Mappings)

The following keywords assign drive mappings.

network_drive=*driveletter=networkpath*

This directive maps the drive *`driveletter`* (do not include the colon) to the network path *`networkpath`* (new in version 1.1).

`networkpath` should be a full network path including the host name and initial backslashes. Note that for this to work, the service must be started using a named user who has the appropriate privileges to access *`networkpath`* (since `LocalSystem) does not have any network privileges`.

local_drive=*driveletter=localpath*

This directive maps the drive *`driveletter`* (do not include the colon) to the local path *`networkpath`* (new in version 1.1).

This is analogous to entering the `SUBST` command at the command line. *`networkpath`* should be a full pathname including the drive letter.

Note that such substitutions are global and immediately visible to other users on the same computer. No special privileges are required (ie `LocalSystem` has sufficient authority to do this).

4.6 Command

The command to run, *`program`* `[` *`program_parameters`* `]` may be any executable program, that is anything with extension `.com`, `.exe` or `.bat`.

`program` and `program_parameters` may refer to environment variables using the `%var%` syntax. These will be substituted (not recursively) where encountered, for example `%HOME%\bin\mycommand.exe %SYSTEMROOT%`.

For command mode, if the -w flag is not supplied, the command can also be a DOS command (eg `dir`).

4.7 Parameters

For command mode, parameters may include substitution text of the form:

```
{prompt}
{prompt:default}
```

Here, **SRVSTART** displays the prompt on `stdout` and reads from `stdin`, substituting the entered text for everything between `{` and `}`.

If the user just presses return, then `default` will be used (if supplied). Either `default` or `prompt` can be empty strings.

If `prompt` or `default` include a space or other symbol meaningful to the NT "shell", surround them in double-quotes.

If prompt begins with a – (hyphen) then the text entered by the user will not be echoed back to the terminal (for passwords etc).

For example:

```
send_server.exe -s{server:MYSERV} -u{user name} -p{-password}
```

This will result in the following interaction:

```
server [MYSERV]: enter server name here or press return for MYSERV
user name: no default
password: response will not be echoed to screen
```

4.8 Install and Remove Modes

You can use **SRVSTART** itself to install or remove services that are based on the `SRVSTART.EXE` executable (new in version 1.1). The syntax is as follows.

1. To install a service which cannot interact with the desktop:

```
srvstart install service_name -c \path\to\controlfile
```

2. To install a service which can interact with the desktop:

```
srvstart install_desktop service_name -c \path\to\controlfile
```

3. To remove a service:

```
srvstart remove service_name
```

The service which is installed has the following characteristics.

- Its short name and display name are both *`service_name`*.
- Its command line is as follows:

```
path\srvstart.exe svc service_name -c \path\to\controlfile
```

where `path\srvstart.exe` is the full path of the **SRVSTART** executable.

- It is set to Manual startup.
- It starts using `LocalSystem` and has no dependencies.
- If the `install_desktop` form is used, then the service can interact with the desktop.

Note that the `remove` option will remove *any* service, not just one installed using **SRVSTART**. It does not prompt for confirmation.

As can be seen this support is quite simple - if you need more powerful management of services then consider using **SVC**.

4.9 Bugs

There is no support at present for pausing or resuming the service. **SRVSTART** will ignore pause or resume requests from the Service Control Manager.

SRVSTART does not extensively validate things supplied to it on the command line or in the control file. A bad invocation can sometimes cause it to core dump.

SRVSTART is pretty relaxed in its error-checking of things like `new`. Failure here should become obvious pretty quickly.

In command mode when the `–w` flag is not supplied, it would be very useful if **SRVSTART** changed the window title (so you could see it in the taskbar). I can't work ou how to do this.

4.10 Version History

Version 1.0 (31 March 2000)

First release version. (Rewritten as a DLL and executable in C++ from `SYBSTART` with some new functionality.)

Version 1.1 (30 June 2000)

- New install functionality (keywords `install`, `install_desktop`, and `remove`).
- `svc` keyword now optional (default).
- Environment substitution applied to all appropriate directives.
- Shut down by sending Windows message (new directive `shutdown_method={kill|command|winmessage}`)
- Truncate log file if first character is > (greater than).
- New directive `drive=`*`x=path`*.
- New directive `localdrive=`*`x=path`*.
- With `auto_restart=y`, does not restart if Windows is shutting down.
- By default, log messages to Event Log if `svc` mode, and to `stdout` if `install`, `install_desktop`, or `remove` mode.
- Version information embedded in **SRVSTART** executables (use `strings`).

5 SRVSTART.DLL - Windows NT Service Support Library

The **SRVSTART** DLL on its own provides all of the service management features of **SRVSTART** to programs written in C++.

This allows you to write your own NT services, making just a few simple calls to manage the service's interactions with the SCM.

5.1 Pre-Requisites

The **SRVSTART** library runs on Windows NT. _95_SUPPORT To run it you require the following files somewhere in your PATH at runtime.

- `srvstart.dll` - **SRVSTART** library
- `logger.dll` - logger library
- `msvcrt.dll` - Microsoft C++ run-time library

At compile/link time, you will need the following:

- `CmdRunner.h` and/or `ScmConnector.h` - **SRVSTART** header files
- `srvstart.lib` - **SRVSTART** library definitions

All of these components are available from my website.

Further information on building programs to use the DLL is given below.

5.2 Synopsis

The **SRVSTART** library exports the following classes.

- `CmdRunner)` (defined in the header file `CmdRunner.h`).
- `ScmConnector` (defined in the header file `ScmConnector.h`).

5.3 Class: CmdRunner

`CmdRunner` is used to start an external program (command or service mode) within a defined environment.

The following examples show how to use `CmdRunner` to run a command in the same or another process.

1. Create a single object instance of the `CmdRunner` class.

```
#include
// typedef enum START_MODES { COMMAND_MODE, SERVICE_MODE, ANY_MODE };
// CmdRunner(START_MODES mode = COMMAND_MODE,char *nm = NULL) throw (SrvStartException);

cmdRunner = new CmdRunner(CmdRunner::COMMAND_MODE,"MY_COMMAND");
```

It is invalid to create more than one `CmdRunner` object.

2. Define the command to run, with any parameters.

```
#include
// void setStartupCommand(const char *sc);
```

```
// void addStartupCommandArgument(const char *arg);

cmdRunner->setStartupCommand("D:\bin\mycommand.exe");
cmdRunner->addStartupCommandArgument("arg1");
// etc.
```

3. (Optional) define any required environment variables.

```
#include
// void addEnv(const char *nm,const char *val) throw (SrvStartException);

cmdRunner->addEnv("MYVAR","MYVALUE");
// etc.
```

4. (Optional) set any other required attributes of the object.

```
#include
// void setStartInNewWindow(bool nw);
// etc.

cmdRunner->setStartInNewWindow(true);
// etc.
```

5. Start.

```
#include
// void start() throw (SrvStartException);

cmdRunner->start();
```

`start()` blocks until the command has completed.

5.4 Class: ScmConnector

`ScmConnector` is used to manage the interaction between a service program and the Windows NT Service Control Manager.

The following examples show how to use `ScmConnector` in a service program.

1. Early on in your program, create a single object instance of the `ScmConnector` class.

```
#include
// ScmConnector(char *svcName,bool allowConnectErrors = false) throw (SrvStartException);

scmConnector = new ScmConnector("MY_SERVICE");
```

It is invalid to create more than one `ScmConnector` object.

2. Install a stop handler. This will be activated by the `ScmConnector` if the service receives a STOP request from the Service Control Manager. The three ways of doing this are described below.

3. Continue starting your service. Once you have completed all of your startup tasks, notify the `ScmConnector` object that your service is running.

```
#include
// void notifyScmStatus(SCM_STATUSES scmStatus,bool ignoreErrors = false) throw (SrvStartExceptio

scmConnector->notifyScmStatus(ScmConnector::STATUS_RUNNING);
```

4. When (if ever) your service is ready to terminate, notify the `ScmConnector` object.

```
#include

scmConnector->notifyScmStatus(ScmConnector::STATUS_STOPPING);
scmConnector->notifyScmStatus(ScmConnector::STATUS_STOPPED);

ExitProcess(...)
```

5.4.1 ScmConnector Stop Handlers

You must install a stop handler for your `ScmConnector` object. This will be activated by the `ScmConnector` if the service receives a STOP request from the Service Control Manager. If you don't do this then it will not be possible for the Service Control Manager to stop your service.

There are three ways to do this.

- Pass the address of a Boolean variable to the `ScmConnector`. If a STOP request is received, the `ScmConnector` will set this variable to `true`).

```
#include
// void installStopCallback(bool *stopRequestedVar) throw (SrvStartException);

bool stopRequested;
scmConnector->installStopCallback(&stopRequested);

 // later ...
if (stopRequested) ...
```

If you select this method, then you will typically poll the variable regularly to see if it has been set to `true`, and if it has, shut down.

- Pass the address of a Win32 event handle (you must create the event yourself). If a STOP request is received, the `ScmConnector` will signal this event.

```
#include
 // void installStopCallback(HANDLE *stopRequestedEvent) throw (SrvStartException);

HANDLE stopEvent = CreateEvent(NULL,FALSE,FALSE,NULL);
scmConnector->installStopCallback(&stopEvent);

 // later ...
WaitForSingleObject(stopEvent,...
```

  If you select this method, you will typically poll the event regularly (or maybe wait on it) to see if it has been signalled, and if it has, shut down.

- Pass the address of a static function. If a STOP request is received, the `ScmConnector` will invoke this function, passing a supplied pointer (which you supply when you install the callback). Note that the function is invoked *asynchronously* in a *separate* thread.

```
#include
 // typedef void STOP_HANDLER_FUNCTION(void*);
 // void installStopCallback(STOP_HANDLER_FUNCTION *stopRequestedFunction,
void *genericPointer) throw (SrvStartException);

 // define the function
void stopCallbackFunction(void *genericPointer) { ... }

scmConnector->installStopCallback(&stopCallbackFunction,
static_cast(this));
```

  If you select this method, your passed function must shut down the service itself.

Whichever method you choose, you must ensure that if the callback is activated, your service program will shut itself down.

5.5 Timing

Timing is crucial with service programs. In particular, the program must connect to the SCM within one second of startup, or Windows NT assumes it is invalid.

To ensure this, you should make sure that you create the `ScmConnector` object early in your program (preferably right at the start). Once you do this, `ScmConnector` will make the appropriate status notifications to the Service Control Manager for you.

Furthermore, a service program must respond promptly to requests from the Service Control Manager (specifically, "interrogate" and "stop" requests). `ScmConnector` handles this for you automatically once it has been instantiated. It runs in its own thread (two threads, actually) and you do not need to take any special actions other than to notify it when your status changes (when you are shutting down).

These constraints aside, there are no restrictions on the time that a service can take to start or stop. However `ScmConnector` will log an informational message if startup takes more than a minute.

5.6 Exception Handling

In general, methods respond to errors by raising a C++ exception. All methods which can throw an exception are defined using the following syntax:

```
class::method(parameters...) throw (SrvStarterException);
```

This syntax is not supported by Microsoft C++ (they claim it is non-standard) but I have included it to denote the methods which can throw exceptions. Note that most constructors fall into this category.

All exceptions are of class `SrvStart::SrvStarterException`, defined in `SrvStart.h`. (`SrvStart` is the namespace here.) Some of the key public data members of this class include:

```
SRVSTART_EXCEPTION exceptionId;
char               className[SRVSTART_EXCEPTION_STRING_SIZE];
char               methodName[SRVSTART_EXCEPTION_STRING_SIZE];
char               errorMessage[SRVSTART_EXCEPTION_STRING_SIZE];
char               sourceFile[SRVSTART_EXCEPTION_STRING_SIZE];
int                lineNumber;
```

Your calling program should catch these exceptions and interrogate the thrown object for further information.

5.7 "Any" Mode

The library implements any mode as follows.

When creating your `ScmConnector` object, set the paramter `allowConnectErrors` to true. Then check the status of the service.

```
#include
 // typedef enum SCM_STATUSES { STATUS_INITIALISING,STATUS_STARTING,
 // STATUS_RUNNING,STATUS_STOPPING,STATUS_STOPPED,
 // STATUS_MUST_START_AS_CONSOLE,STATUS_FAILED };
 // SCM_STATUSES getScmStatus();

ScmConnector::SCM_STATUSES status = scmConnector->getScmStatus();
```

If `status` is `STATUS_MUST_START_AS_CONSOLE` then the `ScmConnector` failed to connect to the Service Control Manager. No exception will be raised in this case.

5.8 Build

The following should be taken into account when building software using the DLL.

- All classes are defined within the namespace `SrvStart`. If you don't want to refer to this namespace directly, then include the following:

```
using namespace SrvStart;
```

- You will need to `#include` the appropriate header file (`CmdRunner.h` or `ScmConnector.h`) for compilation.
- To be compatible with the DLL, you **must** set the `/MD` option for the compiler (Runtime Multithread DLL). From the GUI, select Build|Settings|C++|Code Generation, Use run-time library "Multithread DLL."
- For the linker, you will need to add `srvstart.lib` to the list of library modules.

If you want to include debug code in your service then why not use my `logger.dll`? This provides a simple interface for logging error, information and debug messages to a variety of targets including `stdout`, text files or the Windows NT Event Log. Check out my website for details.

5.9 Bugs

There is no support at present for pausing or resuming the service. `ScmConnector` will ignore pause or resume requests from the Service Control Manager.

5.10 Version History

Version 1.0 (31 March 2000)

First release version. (Rewritten as a DLL and executable in C++ from `SYBSTART` with some new functionality.)

Version 1.1 (30 June 2000)

Enhanced existing classes, and created simple new class `ServiceManager`, to support new functionality described above.

6 Source

SVC and **SRVSTART** are issued under the terms of the GNU General Public License'.

SVC and **SRVSTART** are built using Microsoft Visual C++. I have not used any non-standard features (as far as I am aware) so the source should compile under any C++ compiler for Windows NT. (I haven't tried this.)

Full source code for the executables and DLL is available.

7 Support

I welcome feedback and comments on these programs.

You can email me for support, although I can't guarantee to give any.

Please note that the Sybase features in **SVC** and **SRVSTART** are only included for backwards compatibility and may not be maintained in the future. (I left Sybase several years ago.)

 url: http://rozanski.org.uk/index.php?page=services last changed: 27 November 2011