Topshelf Documentation

Release 3.0

Chris Patterson, Travis Smith, and Dru Sellers

Aug 03, 2017

Contents

Contents:

CHAPTER 1

Topshelf Installation

This section of the online docs will explain how to get Topshelf into your project. It will also show you where to get help, how to report bugs, etc. Hopefully, you will find it useful as you explore the Topshelf framework.

Prerequisites

Topshelf is a .Net framework for C# and will need a .Net runtime to run on.

To work with Topshelf you will need to be running on a Windows operating system. The developers of Topshelf regulary test on Windows 7 and Windows Server 2008RC2. Though it should still work on Windows Server 2003, as long as .Net 3.5 sp1 is installed.

.Net Framework

Currently Topshelf is tested on .NET 3.5 Service Pack 1 and .NET 4.0.

Installing Topshelf

NuGet

The simplest way to install Topshelf into your solution/project is to use NuGet.:

```
nuget Install-Package Topshelf
```

Raw Binaries

If you are a fan of getting the binaries you can get released builds from

http://github.com/topshelf/Topshelf/downloads

Then you will need to add references to

- Topshelf.dll

Compiling From Source

Lastly, if you want to hack on Topshelf or just want to have the actual source code you can clone the source from github.com.

To clone the repository using git try the following:

```
git clone git://github.com/Topshelf/Topshelf.git
```

If you want the development branch (where active development happens):

```
git clone git://github.com/Topshelf/Topshelf.git
git checkout develop
```

Build Dependencies

To compile Topshelf from source you will need the following developer tools installed:

- .Net 4.0 sdk
- ruby v 1.8.7
- gems (rake, albacore)

Compiling

To compile the source code, drop to the command line and type:

```
.\build.bat
```

If you look in the `.\build_output` folder you should see the binaries.

Getting hold of us

Getting hold of the gang behind Topshelf is pretty straightforward. We try to help as much as time permits and have tried to streamline this process as much as possible.

But before you grab hold of us, spend a moment to compose your thoughts and formulate your question. There is nothing as pointless as simply telling us "Topshelf does not work for me," with no further information to give any clue as to why.

And before you even do that, do a couple of searches to see if your question has already been answered. If it has been, you will get your answer much faster that way.

Mailing List

Getting on or off our mailing list happens through Google Groups.

> http://groups.google.com/group/topshelf-discuss/

If you are going to use Topshelf, subscribing to our `topshelf-discuss` mailing list is probably a very good idea. This is where most conversation about Topshelf is going to happen.

Make sure to pick a good subject line, and if the subject of the thread changes, please change the subject to match. Some of us deal with hundreds of emails per day after spam-filters, and we need all the help we can get to pick the interesting ones.

Twitter

The most immediate way to get hold of us is to shoot a tweet to `#tsproj`

Our main time zone is Central time in the United States.

If you can explain your problem in a clear sentence, Twitter is a good way to get quick response. If you do need to paste log files, config and so on, please use a `gist`. But because questions are often not clear without additional context, we may push you to our mailing list on Google Groups.

If Twitter is all quiet, try the mailing list as well. We do have lives, families and jobs to deal with as well.

Issues / Tickets

Please do not open an issue on GitHub unless you have spotted an actual bug in Topshelf. Ask on the mailing list first if you are in doubt.

> https://github.com/topshelf/topshelf/issues

The reason for this policy is to avoid the bugs being drowned in a pile of sensible suggestions for future enhancements, calls for help from people who forget to check back if they get it, and so on.

How to report bugs

If you run into a bug, please spend a minute collecting the right information to help us fix the bug.

The most valuable piece of information you can give us, is always give us a failing unit test, if you can't give us that then how to reproduce the bug in a step by step fashion. Other wise its going to be a lot of back and forth until we can better understand and get to a failing unit test.

CHAPTER 2

Configuring Topshelf

Once Topshelf has been added to your service, you can configure the service using the Topshelf configuration API. Topshelf uses an internal domain specific language (DSL) for configuration, along with a series of fluent builders.

Show me the code!

All right, all right, already. Here you go. Below is a functional setup of Topshelf.

```
1  public class TownCrier
2  {
3      readonly Timer _timer;
4      public TownCrier()
5      {
6          _timer = new Timer(1000) {AutoReset = true};
7          _timer.Elapsed += (sender, eventArgs) => Console.WriteLine("It is {0} and all␣
   ↪is well", DateTime.Now);
8      }
9      public void Start() { _timer.Start(); }
10     public void Stop() { _timer.Stop(); }
11 }
12
13 public class Program
14 {
15     public static void Main()
16     {
17         HostFactory.Run(x =>                                   //1
18         {
19             x.Service<TownCrier>(s =>                          //2
20             {
21                 s.ConstructUsing(name=> new TownCrier());      //3
22                 s.WhenStarted(tc => tc.Start());               //4
23                 s.WhenStopped(tc => tc.Stop());                //5
24             });
25             x.RunAsLocalSystem();                              //6
```

```
26
27              x.SetDescription("Sample Topshelf Host");        //7
28              x.SetDisplayName("Stuff");                       //8
29              x.SetServiceName("Stuff");                       //9
30          });                                                  //10
31      }
32  }
```

Now for the play by play.

1. Here we are setting up the host using the HostFactory.Run the runner. We open up a new lambda where the 'x' in this case exposes all of the host level configuration. Using this approach the command arguments are extracted from environment variables.
2. Here we are telling Topshelf that there is a service of type 'TownCrier". The lambda that gets opened here is exposing the service configuration options through the 's' parameter.
3. This tells Topshelf how to build an instance of the service. Currently we are just going to 'new it up' but we could just as easily pull it from an IoC container with some code that would look something like 'container.GetInstance<TownCrier>()'
4. How does Topshelf start the service
5. How does Topshelf stop the service
6. Here we are setting up the 'run as' and have selected the 'local system'. We can also set up from the command line Interactively with a win from type prompt and we can also just pass in some username/password as string arguments
7. Here we are setting up the description for the winservice to be use in the windows service monitor
8. Here we are setting up the display name for the winservice to be use in the windows service monitor
9. Here we are setting up the service name for the winservice to be use in the windows service monitor
10. Now that the lambda has closed, the configuration will be executed and the host will start running.

Warning: You can only have ONE service! As of 3.x Topshelf the base product no longer support hosting multiple services. This was done because the code to implement was very brittle and hard to debug. We have opted for a simpler and cleaner base product. This feature will most likely come back in the form of an add on nuget.

Topshelf Configuration

While the Quickstart gives you enough to get going, there are many more features available in Topshelf. The following details the configuration options available, and how to use them in Topshelf services.

Service Name

Specify the base name of the service, as it is registered in the services control manager. This setting is optional and by default uses the namespace of the Program.cs file (well, basically, the calling assembly type namespace).

```
HostFactory.New(x =>
{
    x.SetServiceName("MyService");
});
```

It is recommended that service names not contains spaces or other whitespace characters.

Each service on the system must have a unique name. If you need to run multiple instances of the same service, consider using the InstanceName command-line option when registering the service.

Service Description

Specify the description of the service in the services control manager. This is optional and defaults to the service name.

```
HostFactory.New(x =>
{
    x.SetDescription("My First Topshelf Service");
});
```

Display Name

Specify the display name of the service in the services control manager. This is optional and defaults to the service name.

```
HostFactory.New(x =>
{
    x.SetDisplayName("MyService");
});
```

Instance Name

Specify the instance name of the service, which is combined with the base service name and separated by a $. This is optional, and is only added if specified.

```
HostFactory.New(x =>
{
    x.SetInstanceName("MyService");
});
```

This option is typically set using the command-line argument, but it allowed here for completeness.

Service Configuration

The service can be configured in multiple ways, each with different goals. For services that can handle a dependency on Topshelf, the `ServiceControl` interface provides a lot of value for implementing the service control methods. Additionally, a zero-dependency solution is also available when lambda methods can be used to call methods in the service class.

Simple Service

To configure a simple service, the easiest configuration method is available.

```
HostFactory.New(x =>
{
    x.Service<MyService>();
});

// Service implements the ServiceControl methods directly and has a default␣
↪constructor
class MyService : ServiceControl
{}
```

If the service does not have a default constructor, the constructor can be specified, allowing the service to be created by the application, such as when a container needs to be used.

```
HostFactory.New(x =>
{
    x.Service<MyService>(() => ObjectFactory.GetInstance<MyService>());
});

// Service implements the ServiceControl methods directly and has a default␣
↪constructor
class MyService : ServiceControl
{
    public MyService(SomeDependency dependency)
    {}
}
```

If the service needs access to the HostSettings during construction, they are also available as an overload.

```
HostFactory.New(x =>
{
    x.Service<MyService>(hostSettings => new MyService(hostSettings));
});

// Service implements the ServiceControl methods directly and has a default␣
↪constructor
class MyService : ServiceControl
{
    public MyService(HostSettings settings)
    {}
}
```

Custom Service

To configure a completely custom service, such as one that has no dependencies on Topshelf, the following configuration is available.

```
HostFactory.New(x =>
{
    x.Service<MyService>(sc =>
    {
        sc.ConstructUsing(() => new MyService());
```

```
        // the start and stop methods for the service
        sc.WhenStarted(s => s.Start());
        sc.WhenStopped(s => s.Stop());

        // optional pause/continue methods if used
        sc.WhenPaused(s => s.Pause());
        sc.WhenContinued(s => s.Continue());

        // optional, when shutdown is supported
        sc.WhenShutdown(s => s.Shutdown());
    });
});
```

Each of the WhenXxx methods can also take an argument of the `HostControl` interface, which can be used to request the service be stopped, request additional start/stop time, etc.

```
HostFactory.New(x =>
{
    x.Service<MyService>(sc =>
    {
        sc.WhenStarted((s, hostControl) => s.Start(hostControl));
    }
}
```

The `HostControl` interface can be retained and used as the service is running to Stop the service.

Service Start Modes

There are multiple service start modes, each of which can be specified by the configuration. This option is only used if the service is being installed.

```
HostFactory.New(x =>
{
    x.StartAutomatically(); // Start the service automatically
    x.StartAutomaticallyDelayed(); // Automatic (Delayed) -- only available on .NET 4.
↪0 or later
    x.StartManually(); // Start the service manually
    x.Disabled(); // install the service as disabled
});
```

Service Recovery

Topshelf also exposes the options needed to configure the service recovery options as well.

```
HostFactory.New(x =>
{
    x.EnableServiceRecovery(r =>
    {
        //you can have up to three of these
        r.RestartComputer(5, "message");
        r.RestartService(0);
        //the last one will act for all subsequent failures
        r.RunProgram(7, "ping google.com");
```

```
        //should this be true for crashed or non-zero exits
        r.OnCrashOnly();

        //number of days until the error count resets
        r.SetResetPeriod(1);
    });
});
```

Service Identity

Services can be configured to run as a number of different identities, using the configuration option that is most appropriate.

```
HostFactory.New(x =>
{
    x.RunAs("username", "password");
});
```

Runs the service using the specified username and password. This can also be configured using the command-line.

```
HostFactory.New(x =>
{
    x.RunAsPrompt();
});
```

When the service is installed, the installer will prompt for the username/password combination used to launch the service.

```
HostFactory.New(x =>
{
    x.RunAsNetworkService();
});
```

Runs the service using the NETWORK_SERVICE built-in account.

```
HostFactory.New(x =>
{
    x.RunAsLocalSystem();
});
```

Runs the service using the local system account.

```
HostFactory.New(x =>
{
    x.RunAsLocalService();
});
```

Runs the service using the local service account.

Custom Install Actions

These settings allow user-specified code to be executed during the service install/uninstall process.

Before Install Actions

Topshelf allows actions to be specified that are executed before the service is installed. Note that this action is only executed if the service is being installed.

```
HostFactory.New(x =>
{
    x.BeforeInstall(() => { ... });
});
```

After Install Actions

Topshelf allows actions to be specified that are executed after the service is installed. Note that this action is only executed if the service is being installed.

```
HostFactory.New(x =>
{
    x.AfterInstall(() => { ... });
});
```

Before Uninstall Actions

Topshelf allows actions to be specified that are executed before the service is uninstalled. Note that this action is only executed if the service is being uninstalled.

```
HostFactory.New(x =>
{
    x.BeforeUninstall(() => { ... });
});
```

After Uninstall Actions

Topshelf allows actions to be specified that are executed after the service is uninstalled. Note that this action is only executed if the service is being uninstalled.

```
HostFactory.New(x =>
{
    x.AfterUninstall(() => { ... });
});
```

Service Dependencies

Service dependencies can be specified such that the service does not start until the dependent services are started. This is managed by the windows services control manager, and not by Topshelf itself.

```
HostFactory.New(x =>
{
    x.DependsOn("SomeOtherService");
});
```

There are a number of built-in extension methods for well-known services, including:

```
HostFactory.New(x =>
{
    x.DependsOnMsmq(); // Microsoft Message Queueing
    x.DependsOnMsSql(); // Microsoft SQL Server
    x.DependsOnEventLog(); // Windows Event Log
    x.DependsOnIis(); // Internet Information Server
});
```

Advanced Settings

EnablePauseAndContinue

Specifies that the service supports pause and continue, allowing the services control manager to pass pause and continue commands to the service.

```
HostFactory.New(x =>
{
    x.EnablePauseAndContinue();
});
```

EnableShutdown

Specifies that the service supports the shutdown service command, allowing the services control manager to quickly shutdown the service.

```
HostFactory.New(x =>
{
    x.EnableShutdown();
});
```

OnException

Provides a callback for exceptions that are thrown while the service is running. This callback is not a handler, and will not affect the default exception handling that Topshelf already provides. It is intended to provide visibility into thrown exceptions for triggering external actions, logging, etc.

```
HostFactory.New(x =>
{
    x.OnException(ex =>
    {
        // Do something with the exception
    });
});
```

Service Recovery

To configure the service recovery options, a configurator is available to specify one or more service recovery actions. The recovery options are only used when installing the service, and are set once the service has been successfully

installed.

```
HostFactory.New(x =>
{
    x.EnableServiceRecovery(rc =>
    {
        rc.RestartService(1); // restart the service after 1 minute
        rc.RestartSystem(1, "System is restarting!"); // restart the system after 1
↪minute
        rc.RunProgram(1, "notepad.exe"); // run a program
        rc.SetResetPeriod(1); // set the reset interval to one day
    })
});
```

The recovery actions are executed in the order specified, with the next action being executed after the previous action was run and the service failed again. There is a limit (based on the OS) of how many actions can be executed, and is typically 2-3 actions.

Logging Integration with Topshelf

By default, Topshelf uses a TraceSource for logging. This is part of the .NET framework, and thus does not introduce any additional dependencies. However, many applications use more advanced logging libraries, such as Logary or NLog. To support this, an extensible logging interface is used by Topshelf.

Logary integration

To ship logs with Logary, use the Logary.Adapters.Topshelf nuget. Once you've added this nuget to your project, you can configure Topshelf to use it via the configuration builder:

```
using (var logary = ...Result)
    HostFactory.New(x =>
    {
        x.UseLogary(logary);
    });
```

This makes it possible to get your logs off your node, so that you avoid running out of disk space and can log to modern log tagrets, such as ElasticSearch and InfluxDB.

For more information, see the Logary README at https://github.com/logary/logary.

log4net Integration

To enable logging via log4net, the Topshelf.Log4Net NuGet package is available. Once added to your project, configure Topshelf to use log4net via the configuration:

```
HostFactory.New(x =>
{
    x.UseLog4Net();
});
```

This will change the `HostLogger` to use log4net. There is an overload that allows a configuration file to be specified. If given, the filename will be resolved to the ApplicationBase folder and passed to log4net to configure the log appenders and levels.

NLog Integration

To enable logging via NLog, the Topshelf.NLog NuGet package is available. Once added to your project, configure Topshelf to use NLog via the configuration:

```
HostFactory.New(x =>
{
    x.UseNLog();
});
```

This will change the `HostLogger` to use NLog. An existing LogFactory can be passed as well, using an overload of the same method.

CHAPTER 3

Topshelf Overview

Topshelf is a framework for hosting services written using the .NET framework. The creation of services is simplified, allowing developers to create a simple console application that can be installed as a service using Topshelf. The reason for this is simple: It is far easier to debug a console application than a service. And once the application is tested and ready for production, Topshelf makes it easy to install the application as a service.

Topshelf Key Concepts

Why would I want to use Topshelf?

Topshelf is a Windows service framework for the .NET platform. Topshelf makes it easy to create a Windows service, test the service, debug the service, and ultimately install it into the Windows Service Control Manager (SCM). Topshelf does this by allowing developers to focus on service logic instead of the details of interacting with the built-in service support in the .NET framework. Developers don't need to understand the complex details of service classes, perform installation via InstallUtil, or learn how to attach the debugger to services for troubleshooting issues.

How does Topshelf do this?

When a developer uses Topshelf, creating a Windows service is as easy as creating a console application. Once the console application is created, the developer creates a single service class that has public Start and Stop methods. With a few lines of configuration using Topshelf's configuration API, the developer has a complete Windows service that can be debugged using the debugger (yes, F5 debugging support for services) and installed using the Topshelf command-line support.

What if my service requires custom installation options?

Topshelf supports most of the commonly used service installation options, including:

- Automatic, Automatic (Delayed), Manual, and Disabled start options.

- Local System, Local Service, Network Service, Username/Password, or prompted service credentials during installation.
- Service start dependencies, including SQL Server, MSMQ, and others.
- Multiple instance service installation with distinct service names.
- Service Recovery options, including restart, reboot, or run program.

What's the impact to my service?

Topshelf is a small (around 200 KB) assembly with no dependencies, making it easy to integrate with any service.

What about command-line arguments?

Topshelf has an extensible command-line, allowing services to register parameters that can be specified using command-line arguments.

What else can I get by using Topshelf?

Topshelf is an open-source project, and new contributions are being accepted regularly. While the base Topshelf assembly is being kept small with very specific functionality, additional assemblies that build on top of Topshelf are also being created. For example, a supervisory extension that monitors service conditions such as CPU and memory is being created that will automatically cycle the service if conditions exceed specifications. This reduces the need to closely monitor services and manually restart them when they misbehave.

Is Topshelf just for Windows?

Topshelf works with Mono, making it possible to deploy services to Linux. The service installation features are currently Windows only, but others are working on creating native host environment support so that installation and management features are available as well.

Topshelf Command-Line Reference

Once a service has been created using Topshelf, an extensive command-line vocabulary is available which can be used to install, uninstall, start, and configure the service.

The command-line help can be displayed at any time by entering `myService.exe help` on the command-line.

Help Text

The help text from the command line is shown below for easy reference.

service.exe **[verb]** [-option:value] [-switch]

run Runs the service from the command line (default)

help or **–help** Displays help

install Installs the service

-username	The username to run the service

-password	The password for the specified username
-instance	An instance name if registering the service multiple times
--autostart	The service should start automatically (default)
--disabled	The service should be set to disabled
--manual	The service should be started manually
--delayed	The service should start automatically (delayed)
--localsystem	Run the service with the local system account
--localservice	Run the service with the local service account
--networkservice	Run the service with the network service permission
--interactive	The service will prompt the user at installation for the service credentials
--sudo	Prompts for UAC if running on Vista/W7/2008
-servicename	The name that the service should use when installing
-description	The service description the service should use when installing. Eg: -description: MyService Eg: -description "My Service"
-displayname	The display name the the service should use when installing Eg: -displayname: MyService Eg: -displayname "My Service"

start Starts the service if it is not already running

-instance	The instance to start

stop Stops the service if it is running

-instance	The instance to stop

uninstall Uninstalls the service

-instance	An instance name if registering the service multiple times
--sudo	Prompts for UAC if running on Vista/W7/2008

Examples

Basic Service Installation

MyService.exe install -username:DOMAINServiceAccount -password:itsASecret -servicename:AwesomeService –autostart

Service Installation with Quoted Arguments

MyService.exe install -username "DOMAINService Account" -password:"Its A Secret" -servicename "Awesome Service" –autostart

CHAPTER 4

Troubleshooting Topshelf

CHAPTER 5

Indices and tables

- genindex
- modindex
- search