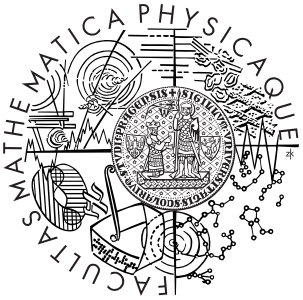
Charles University in Prague
Faculty of Mathematics and Physics

SOFTWARE PROJECT



«XRouter»

6. DaemonNT

Soběslav Benda
Miroslav Cicko
Tomáš Kroupa
Petr Sobotka
Bohumír Zámečník

Supervisor: Mgr. Martin Nečaský, Ph.D.

2011

DaemonNT – documentation

Intended audience: system administrators, service developers.

Concepts

Introduction

DaemonNT provides a environment for hosting programs as Windows services. It can install services and control their running. DaemonNT is designed to be easy to configure, use and extend. In addition, its offers its own simple, configurable, yet high-performance logging facility and enables developers with a debugging mode. So that DaemonNT is useful not only for deployment of services but also for their development.

Windows service [5] (formerly known as NT service) is a long-running program which is started automatically at system boot and runs on background without direct user intervention (eg. via GUI). It is a concept similar to Unix daemons. The name DaemonNT is inspired from this. An example of a Windows service can be MS SQL Server or Internet Information Server.

In context of the XRouter project, DaemonNT is an independent project serving as a foundation for running various XRouter services in a unified way. It is implemented in C#/.NET and its only external dependency is SchemaTron library (for configuration files). Although DaemonNT implementation uses Windows NT services and currently tied to Windows/.NET platform, it provides an abstraction for service hosting and can be theoretically implemented also on different platforms.

Features

- hosting of programs as services
    - running a service as Windows service or in debug mode
        - Debug mode enables running the service in a console application without the need for being installed as a Windows service. This simplifies development.
    - built-in installer/uninstaller for each service
        - No external tools are needed for deploying the service as a Windows service. DaemonNT has a built-in installer for each service it manages.
- configuration via a XML file or a GUI tool
    - comprehensible configuation of each service (all DaemonNT services can be configured at one place)
    - possibility to configure and run multiple instances of a single program
        - it is then possible to run multiple instances of the same DaemonNT service (on one OS instance)
    - GUI editor of the configuration files
- command-line tool with all necessary commands
- logger facility
    - can be used from the hosted services
    - provides two different log types: event log, trace log
    - high-performance
    - logging from multiple threads possible
    - configurable and extendable log storages
        - log to files or elsewhere
- free open-source software

Motivation – Why yet another framework?

As the primary user of DaemonNT is the XRouter project, DaemonNT is designed with its needs in mind. XRouter is a server system, so it is natural to host its components within Windows services.

The .NET framework provides a nice API for using the Windows service framework. Its usage is made more easier compared to the native API, still it is not completely straightforward and high-level, especially for the installers. Also it is not suitable to tightly couple XRouter components with their service hosting. Thus an isolated abstraction was wanted instead directly using the .NET Windows service API from XRouter.

It is worth mentioning that Windows provides nice standard tools for controling Windows services and also an event log API and viewer tool. With DaemonNT those tool can be used as usual.

A very important design goal was to allow easy but powerful unified configuration of all hosted services, at best using XML files. Also XRouter needed better logging facility than standard windows log.

Project information

DaemonNT is released as open-source software. In context of XRouter it provides a unified easy-to-use platform for hosting the XRouter service. However, from the technical view it is independent on XRouter and can be used for different projects. Currently it shares the project hosting, including the Git repository, with the XRouter project.

Licence

DaemonNT is released under the terms of the MIT License. See the LICENSE file.

Authors

- Soběslav Benda – design, programming, documentation
- Bohumír Zámečník – unit tests, code revisions, documentation, programming

Source code

Source codes are available in the main Git repository hosted at Assembla

- Code browser: http://www.assembla.com/code/xrouter/git/nodes
- Git repository clone URL (read-only): git://git.assembla.com/xrouter.git

Downloads

Latest binary releases are available at Assembla:

- https://www.assembla.com/spaces/xrouter/documents/tag/release

Bug tracking

You can report problems or request features in a ticketing system hosted at Assembla:

- http://www.assembla.com/spaces/xrouter/tickets

User documentation

System requirements

- Windows NT-family operating system (Windows XP, 7, Server 2008, …)

- Microsoft.NET Framework 4.0

Configuration

Assuming we have a DaemonNT service prepared in an assembly and want to to run it, first the service must be described in a configuration file. Typically all services managed by DaemonNT are configured in a single file, by default named `DaemonNT.xml` and located in the same directory as the `DaemonNT.exe` executable. The configuration file is in XML format described by a Schematron schema. In case several separate configuration files are needed it is possible to specify another file.

A sample configuration file might look like this:

```
<config>
 <service type="DaemonNT.Example.ExampleService,DaemonNT.Example.dll"
   name="ExampleServer">
  <installer>
   <description>Description of an example service.</description>
   <start-type value="Automatic"/>
   <account value="LocalSystem"></account>
  </installer>
  <trace-logger buffer-size="5000">
   <storage type="DaemonNT.Example.TestTraceLogger,DaemonNT.Example.dll"
     name="ConsoleStorage"/>
  </trace-logger>
  <settings>
   <section name="timer">
     <param name="interval">1000</param>
   </section>
   <param name="xxx">Hello World!</param>
  </settings>
</service>
</config>
```

The meaning of the most important XML elements is as follows:

- `<config/>` = the root element containing an arbitrary number of `<service/>` elements
- `<service/>` = specification of a single DaemonNT service instance
    - the `type` atributte contains the name of class which implements the service and path to the respective assembly
    - the `name` atributte contains the identifier of this service instance (unique among all installed Windows services in the operating system)
    - the `<service/>` element can then contain the following optional elements
- `<installer/>` = service installer parameters
- `<trace-logger/>` = settings of the trace-logger provided by DeamonNT
- `<settings/>` = user-defined hierarchical parameter settings. Those settings have a general tree structure which enables users to better structure key-value parameters into sections – `<section/>`. The `<settings/>` element can also be though of as the top-level section.

Each configuration element is described in more detail further at each use-case.

Usage

Command-line tool – DaemonNT.exe

For controlling the services there is a command-line tool `DaemonNT.exe`. In can be run with parameters of the following syntax:

```
DaemonNT.exe [options] <command> [service-name]
<command> ::= debug | run | install | uninstall | start | stop | restart | status
| list
```

The tool provides several commands, one of which is selected by the mandatory *<command>* parameter. All commands except `list` operate with a single service, which is specified in the *service-name* parameter. Is a OS-wide unique identifier of the service and corresponds to the `service.@name` attribute in the configuration file. This parameter is mandatory for all commands except `list`.

The `[options]` section represent any optional parameters. Currently there is one available:

- `--config-file=`*<config-file>* specifies a path to a configuration file to be used instead of the default one. *<config-file>* represents an absolute or relative path to the configuration file. Relative path is related to the directory containing the `DaemonNT.exe` program (not the current working directory!). The default configuration file is `DaemonNT.xml`.
- `-w` indicates that the process should wait for a key press just before exitting in order to keep the console open. This can be useful when running the command from a shortcut instead of a console.

Debug

The debug command runs the service in the DaemonNT's debug mode – the service is ran as an ordinary user console application instead of a Windows service. This mode is suitable for developers to shorten the edit-compile-run cycle. In this case the service needn't be installed to the operating system, thus Administrator rights are not needed. Moreover this way it is possible to run tools like debugger, profiler, code coverage, which is almost impossible or at least very hard when running the program as a Windows service.

In order to run a service in debug mode it has to be specified in the configuration file.

A minimal example:

```
<service type="class, assembly" name="MyServer"/>
```

The service ran this way can be stopped by `CTRL+C` keys in its console window.

Run

Runs a configured and correctly installed service as a Windows service. Only the operating system can run this command (eg. via manual service start in the Windows Services window)! In case a user tries to start a service with this command (eg. from a console) the operating system shows an error message and does not allow starting the service this way.

Install

DaemonNT has a built-in Windows service installer so there is no need for an external tool, such as `installutil.exe`. In order to install a service it has to be specified in the configuration file.

For example:

```
<service type="class, assembly" name="MyServer"/>
```

If the default installer settings are not alright it is possible to override each of its parameters in the `<installer/>` inside the `<service/>` element. All parameters are optional, so you can change only some of them.

Example:

```
<installer>
   <description>A service description</description>
   <start-type value="Automatic"/>
   <account value="User">
     <username>John Doe</username>
     <password>secretpassword</password>
   </account>
   <depended-on>Service1, Service2</depended-on>
</installer>
```

The meaning of the parameters is as follows:

- `<description/>` = a text briefly describing what is the purpose of the service (empty string by default). It is displayed in the standard Windows Services GUI.
- `<start-type/>` = specified how the service should be started (manually, automatically at system boot). Its `value` attribute might contain one of the following value: `Automatic`, `Manual`, nebo `Disabled`. With last value the service cannot be started which might be sometimes useful. The default value is `Manual`.
- `<account/>` = the account type the service runs with. The `value` atributte might contain the following values: `LocalSystem`, `LocalService`, `NetworkService`, or `User`. Their meaning is described in the MSDN documentation of the System.ServiceProcess.ServiceAccount [6] type. If the `User` value is set then it is needed also to specify its subelements `<username/>` and `<password/>`. The default value of the `<account/>` element is `LocalSystem`.
- `<depended-on/>` = the list of any Windows services (not only DaemonNT services) required for running this service (ie. services which must run before this service can be started). When starting this service all required services which do not run are started automatically and only then this service is started. This element contains a comma-separated list of Windows service identifiers of required services. Any spaces are ignored. The default value is an empty list.

After configuring the service and its installer it is possible to install the service. When installing on newer Windows (newer than XP, eg. Windows 7) there are some additional security policies requiring the installer to be run with Administrator privileges.

Example of installing a service named `MyServer` in Windows 7:

- `run cmd.exe` with the Administrator privileges ("Run as administrator")
- `DaemonNT.exe install MyServer`

This command first loads and validates the configuration file and configures the built-in installer. Then a transactional installation is performed. In case of error the installation rolls back to the original state. The process of installation is logged into a file named like *`service-`*

name.Installer.log to the directory where DaemonNT.exe executable is located.

Please note that the installer only registers the command to run the service. No files are copied and no instance of the installed service is created yet. This means that the service configuration beyond the installer section (eg. service type and assembly) might not be specified yet or might be changed later without trouble.

After a successful installation the service should be visible in the Windows Services GUI (Windows 7: *Control Panel -> Administrative Tools -> Services*). The Windows service display name for any DaemonNT service consists of the service identifier prefixed with DaemonNT in order to clearly determine which Windows services are managed by DaemonNT. For example you can find a service named MyServer as DaemonNT MyServer.

Uninstall
Uninstalls a previously installed service. No configuration file is checked, only the uninstaller is ran with a specified service identifier. The installer creates or updates the *service-name*.Installer.log file (located in the same directory as directory as the DaemonNT.exe executable).

Having the Windows Services GUI open while the uninstaller is ran the service does not vanish from the list but seems to be disabled. The correct status can be seen after this windows is updated with F5 key or closed and opened again.

Start
Starts the requested service (if it is not running). The result is the same as starting it from the Windows Services GUI or by typing the command net start *service-name*. Administrator privileges might be needed to start the service successfully. To do this the console has to be started using "Run as Administrator".

Stop
Stops the requested service (if it is running). The result is the same as stopping it from the Windows Services GUI or by typing the command net stop *service-name*. Administrator privileges might be needed to start the service successfully.

Restart
Restarts the requested service. The result is the same as stopping it and then starting it again using any previously mentioned method. Administrator privileges might be needed to do this successfully.

Status
Show the status of running of the requested service, eg. Runnning, Stopped, etc. If the service is not installed the program informs about it.

Example session with the above commands:

```
> DaemonNT status MyServer
Service MyServer is not installed.
> DaemonNT install MyServer
> DaemonNT status MyServer
Status of service MyServer: Stopped.
> DaemonNT start MyServer
> DaemonNT status MyServer
```

```
Status of service MyServer: Running.
> DaemonNT stop MyServer
> DaemonNT status MyServer
Status of service MyServer: Stopped.
```

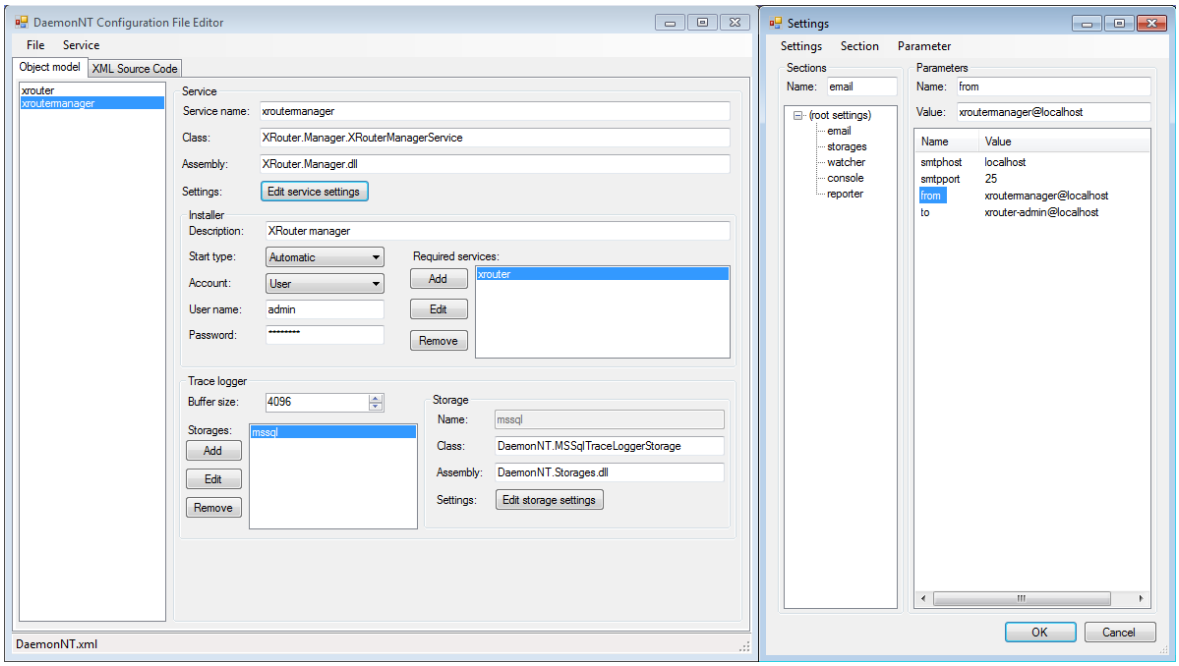
List

Show the list of all DaemonNT services configured in the current configuration file. In conrast to the other commands the parameter *`service-name`* is not used here.

Configuration editor GUI – DaemonNT.GUI.ConfigEditor.exe

Configuration files can be visually edited in the provided editor. This is useful especially for non-expert user trying to configure DaemonNT for the first time or just to make some simple changes without looking in the manual for the exact syntax.

The resulting XML code can be displayed immediately and can be validated using the built-in Schematron schema.



Logs

DaemonNT might log into four types of logs:

- event log
- trace log
- installer log
- system event log

A running service can utilize DaemonNT's own event log and trace log. There is an efficient multi-threading logger available, which is based on the producers-consumer pattern. It is possible to log from multiple thread simulataneously while the logs are received and stored by a single logger thread.

The target audience of event logs are system administrators and they are supposed to be rather sparse and of a small amount. The purpose is to log important events during the production run-time of DaemonNT services.

On the other hand, trace logs are useful rather to get a detailed insight in case of tracing

a problem – while debugging during development or in production. They should provide more detailed information specific to the problem domain. Also they can be structured (eg. using XML) to ease automated processing as they can be larger in volume. There is a nice comparison of the meaning of logging and tracing [7].

During the service installation or uninstallation the installer logs into its own log file. Finally, the standard Windows Service Event Log is used only in a minimalistic way. Besides the service start and stop events only severe error are logged, especially when there are no other log facilities available. (Eg. the service has not the right to write logs to a directory.)

Log files are store in a path relative to the `DaemonNT.exe` executable:

- event logger files – `Logs/`*`YYYY_MM_DD_service-name`*`.log`
- trace logger files – `Logs/`*`YYYY_MM_DD_service-name`*`.Trace.log`
- (un)installer log files – *`service-name`*`.Installer.log`
- Windows Event Log – it has its own storage completely outside the scope of DaemonNT

An example of event log records:

```
21:26:14.32    I    The 'HeavilyLoggingService' service is being started...
21:26:14.58    I    ServiceName=HeavilyLoggingService IsDebugMode=True
21:26:14.65    I    Logged 1000 trace log event in 60 ms
21:26:14.66    I    The 'HeavilyLoggingService' service has beed started
successfully!
21:27:31.73    I    The service is being stopped...
21:27:31.73    I    The service has been stopped successfully!
```

An example of trace log records (notice the XML structure):

```
<log date-time="2011-05-09T16:28:49.48" type="I" thread-id="5">tick</log>
<log date-time="2011-05-09T16:28:49.48" type="I" thread-id="5"><date-time
date="2011-05-09"><hour>16</hour><min>28</min><sec>49</sec></date-time></log>
```

Programmer documentation

This section is intended for programmers who need to maintain or extend the DaemonNT project itself, as well as for programmers writing new services to be hosted within DaemonNT.

Infrastructure

Let us describe how DaemonNT is divided into several parts corresponding to namespaces. The goal is mainly to introduce a programmer into DaemonNT implementation. For more details on API and implementation please refer to the Doxygen documentation.

Namespace DaemonNT

It contains the implementation of the core commands of DaemonNT (such as starting/ stopping services) available as an API (`DaemonNT.ServiceCommands`) and also a console tool for invoking those commands.

Each DaemonNT service have to be derived from the abstract class `DaemonNT.Service`. Such a service can then be hosted either in the run-time (Windows service) host or the debug host.

The `ServiceRuntimeHost` class, derived from `System.ServiceProcess.ServiceBase`, represents a Windows service within which the DaemonNT service can run. This host is created when running the service by the system's Service Control Manager. In contrast, the `ServiceDebugHost` class wraps the DaemonNT service, so that it runs as a user-space console application.

The `DaemonNT.exe` console tool, implemented in the `DaemonNT.Program` class, just parses the command-line arguments and the configuration file and runs the appropriate commands from the `ServiceCommands` API (eg. installing or running a service).

Namespace DaemonNT.Configuration

Provides the support to configure DaemonNT services. It can be represented both by a provided object model and by XML. There are means for conversion in both directions and also a configuration loader.

Namespace DaemonNT.Installation

It provides an embedded NT service installer and some related operations, such as installing or querying for installed services. The features are very similar to the `InstallUtil.exe` installation utility [4] shipped with the .NET Framework.

Namespace DaemonNT.Logging

Provides a simple logging facility with a separated front-end and storage(s). There are two log types – event log and trace logs (each with a slightly different purpose). In addition to the built-in file storage various custom storages can be implemented as extension to DaemonNT. It is sufficient just to implement the `ILoggerStorage` inteface and register the storage with a service.

Namespace DaemonNT.Resources

Provides some non-code resources, such as the Schematron schema for validating the configuration XML format.

Extensibility and its examples

In the following text you may find the information and examples of the ways how to extend the DaemonNT: create and configure a new service, create a new trace logger storage. The projects might be developed in any .NET language, the examples below are only in C#, though.

When creating a new project for an extension you have to add DaemonNT as a reference: either add `DaemonNT.csproj` as a Project reference or add `DaemonNT.exe` assembly as a File reference.

DaemonNT provides some base code for every service or trace logger storage and extension classes have to be inherited, so in this respect DaemonNT can be thought as a heavy-weight framework (compared to freely implementing interfaces without deriving classes).

New service implementation and configuration

Implementation

Each DaemonNT service is represented by a class derived from `DaemonNT.Service`. The important thing is to override the hook methods `Service.OnStart()` and `Service.OnStop()` with the custom code to start and stop the service.

The typical usage scenario is that the service consists of an infinite event processing loop running in its own thread. The `OnStart()` method launches the thread and `OnStop()` aborts it. Note that the `OnStart()` is completely not suitable to loop inside this method as it is supposed just to quickly initialize the service and return. The following example show a minimal code to run a service in a separate thread. For simplicity it uses interactive Console for interaction which is only available in debug mode - there is no Console available for a Windows Service. Typically services communicate over network.

```
public class ExampleService : Service {

 Thread thread;

 protected override void OnStart(OnStartServiceArgs args) {
   // initialize and start the service thread
   thread = new Thread(new ThreadStart(this.ServiceLoop));
   thread.Start();
 }

 private void ServiceLoop() {
   try {
     while (true) {
       string request = Console.ReadLine();
       Console.WriteLine(request.ToLower());
     }
   } catch (ThreadAbortException ex) {
     Console.WriteLine("Service loop thread was aborted.");
   }
 }

 protected override void OnStop(OnStopServiceArgs args) {
   thread.Abort();
 }
}
```

A complete example of a custom DaemonNT service is available in the DaemonNT repository in the `DaemonNT\DaemonNT.Test\TickingService.cs` file.

Configuration

To make a service known to DaemonNT it must be registered in the configuration file. This connects a service identifier with a particular type located in an assembly creating a service instance. One service type can be run in multiple instances, each potentially with a different configuration. A minimal service configuration looks like this:

```
<service type="DaemonNT.Test.ExampleService, DaemonNT.Test.dll"
name="ExampleServer" />
```

In order to provide the service with some parameters each service instance can have its settings – a collection of key-value pairs hierarchically structured into sections. Both parameter names and values are just strings. Conversion to any other types (numbers etc.)

is up to service developers.

Example of a service with settings:

```
<service type="DaemonNT.Test.ExampleService, DaemonNT.Test.dll"
 name="ExampleServer">
  <settings>
   <section name="timer">
     <section name="Inner">
       <param name="interval">1000</param>
     </section>
     <param name="InnerSectionParam">Hello World!</param>
   </section>
   <param name="MeaningOfLife">42</param>
  </settings>
<service/>
```

Those settings are available in the `OnStart()` method argument, already parsed into a dictionary-like structure.

Example:

```
protected override void OnStart(OnStartServiceArgs args)
{
 string answer = args.Settings.Parameter["MeaningOfLife"];
 var intervalStr = args.Settings["timer"]["Inner"].Parameter["interval"];
 int interval = Convert.ToInt32(intervalStr);
}
```

New trace logger storage

By default, trace logger stores its records into one storage made of file(s) in some format. In case the log records should also go to a different storage (eg. log server, database, different file format) we need to create a custom trace logger storage and add it to the existing one. Note: currently it is not possible to remove or disable the default storage.

Implementation

The custom trace logger storage should be a class derived from `DaemonNT.TraceLoggerStorage` and implement the `OnSaveLog()`.

Example:

```
protected override void OnSaveLog(TraceLog log) {
 if (this.isDebugMode) {
     Console.WriteLine(String.Format("{0}: {1}",
       log.DateTime.ToString("HH-mm-ss"), log.Content));
 }
}
```

A full example of a custom trace logger storage which prints records at the Console is in the DaemonNT repository in the `DaemonNT\DaemonNT.Test\ConsoleTraceLoggerStorage.cs` file.

Configuration

When the custom storage is done it can be attached to a service instance in the configuration file. Trace logger settings for a particular service instance are located inside the `<trace-logger/>` element within `<service/>`. Custom trace logger storage is described in the `<storage/>` element. It contains storage identifier, its type and assembly and storage's own settings.

Example service with a custom trace logger storage added:

```
<service name="ServiceWithMultipleStorages"
 type="DaemonNT.Test.ServiceWithMultipleStorages, DaemonNT.Test.dll">
 <trace-logger buffer-size="1024">
   <storage name="ConsoleStorage"
     type="DaemonNT.Test.ConsoleTraceLoggerStorage, DaemonNT.Test.dll">
     <settings />
   </storage>
 </trace-logger>
</service>
```

References

[1] XRouter - http://www.assembla.com/spaces/xrouter
[2] Schematron language - www.schematron.com
[3] SchemaTron validator library - Google docs
[4] InstallUtil.exe – http://msdn.microsoft.com/en-us/library/50614e95(v=vs.80).aspx
[5] Windows Service – http://en.wikipedia.org/wiki/Windows_service
[6] ServiceAccount enum – http://msdn.microsoft.com/en-us/library/system.serviceprocess.serviceaccount.aspx
[7] Tracing – http://en.wikipedia.org/wiki/Tracing_%28software%29