

HOWTO: Compile Java applications to NT services

Article ID: 000019

Last Revised On: 26-Dec-2005

The information in this article applies to:

- Excelsior JET for Windows, Professional Edition version 3.0 and above

All mentioned examples for this article may be found [here](#).

1 OVERVIEW

A Windows service, also known as NT service, is a special long-running process that may be launched during operating system bootstrap. An essential feature of a service is the ability to run even if no user is logged on to the system. Examples of services are FTP/HTTP servers, print spoolers, file sharing, etc. Services are supported only in the Microsoft Windows NT/2000/XP/2003 operating system product line (Windows 95/98/ME do not support services).

Typically, Windows services have not a user interface but are managed through the Services applet of the Windows Control Panel, or a separate application or applet. Using the standard Services applet, a user can start/stop, and, optionally, pause/continue a previously installed service. The common way for a service to report a warning or error is recording an event into the system event log. The log can be inspected using the Event Viewer from Administrative Tools.

A service program is a conventional Windows executable associated with a unique system name using which it can be installed to/removed from the system. A service can be marked for automatic startup (i.e. to be launched at system bootstrap) or manual startup (to be activated by another service or by a user through Control Panel/Services).

2 DETAILS

Excelsior JET eases conversion of programs written in the Java programming language into Windows services through providing the Excelsior WinService API and supporting it in the compiler and runtime. The API has one base class `WinService` with special methods called *control handlers*. To convert your Java application to a Windows service, you must extend the `WinService` class with your own and implement one or more control handler methods in it.

For instance, when the user selects your service in the Control Panel/Services applet and clicks the **Pause** button, the `pause()` method of your `WinService` subclass is called. You should implement it to provide the pause functionality.

The base class `WinService` implements all handler methods as doing nothing, so if your service does not support pause/resume and does not need any initialization or cleanup on start/stop/shutdown, you only have to override the `WinService.run()` method, implementing the main functionality of your service.

2.1 An example

Suppose you have a simple Java program which you want to convert to an NT Service:

```
import java.io.*;

public class HelloWorld {
    public static void main (String[] args) {
        for(;;) {
            System.out.println ("Hello, world!");

            try { Thread.sleep (60*60*1000); }
            catch(InterruptedException t) {}
        }
    }
}
```

2.1.1 Converting the Java program to an NT service

First, make the `HelloWorld` class extending `com.excelsior.service.WinService` class. Then, services have no single entry point such as the `main` method in conventional Java applications. Instead, they must implement control handler methods of the base class `WinService`. The main body of the service should be placed in the `run` handler.

Note, that the instance of the `HelloWorld` class is created implicitly by the JET runtime during the service startup. So ensure that the default constructor is declared in the service class, and has the `public` access modifier.

Finally, services do not have a console or standard output stream. The common way for a Windows service to issue a message is to add an entry to the system event log, which can be browsed using Event Viewer from Administrative Tools. So it is necessary to replace the `System.out.println()` call with a call of `WinService.logInfoEvent()`.

```
public class HelloWorld extends com.excelsior.service.WinService {
    public HelloWorld() {
    }

    public void run () {
        for(;;) {
            logInfoEvent ("Hello, world!");

            try { Thread.sleep (60*60*1000); }
            catch(InterruptedException t) {}
        }
    }
}
```

This service simply adds an informational event with the text "Hello, world!" to the system event log every hour. For more information about writing to the system event log, see *Excelsior JET for Windows User's Guide* (Chapter "Windows NT services", Sections "Windows services overview" and "Excelsior WinService API").

IMPORTANT: the `run()` method implements the main service functionality. Upon return from that method, the service stops. So the `run()` method is very similar to a conventional Java program's `main()` method, with one major distinction:

- A conventional Java application will continue working even after the `main()` method has finished, until all non-daemonic threads exit.
- When the `run()` method of a `WinService`-based application ends, other threads, if any, are stopped immediately.

If you wish your service program to wait for a proper exit of other threads, you should implement that programmatically in the `run()` method.

2.1.2 Adding initialization

You may separate service initialization from its main functionality by moving initialization code to the `init()` method, which is invoked before the `run()` method when the service is started. The service reports to the system that it is up and running by completing the `init()` method.

The `init()` method returns a boolean indicating whether the service accepts pause/resume control requests.

If the `init()` method has completed in time (see below), the `run()` method is invoked immediately.

```
import java.io.*;

public class HelloWorld extends com.excelsior.service.WinService {
    private volatile boolean paused = false;

    public HelloWorld() {
    }

    public void run () {
        for(;;) {
            logInfoEvent ("Hello, world!");

            try { Thread.sleep (15000); }
            catch(InterruptedException t) {}
        }
    }

    public boolean init () {
        // things to do before service runs

        logInfoEvent("init() method called, initializing HelloWorld service...");

        // no suport for pause/resume
        return false;
    }
}
```

2.1.3 Implementation of service controls

To allow the user to control the service, implement the respective control methods (if your service supports pause/resume requests, make sure that the `init()` handler returns `true`.)

```
import java.io.*;

public class HelloWorld extends com.excelsior.service.WinService {
    private volatile boolean paused = false;
    private Object pauseLock = new Object();

    public HelloWorld() {
    }

    public void run () {
        for(;;) {
            if(paused) {
```

```

        synchronized(pauseLock) {
            try { pauseLock.wait(); }
            catch(InterruptedException t) {}
        }
    }

    logInfoEvent ("Hello, world!");

    try { Thread.sleep (60*60*1000); }
    catch(InterruptedException t) {}
}

public boolean init () {
    // things to do before service run

    logInfoEvent("init() method called, initializing service...");

    return true;
}

public void shutdown () {
    // things to do before system shutdown
}

public void stop () {
    // things to do before the service stops

    logInfoEvent("stop() method called, service is stopped");
}

public void pause () {
    // things to do on service pause request

    logInfoEvent("pause() method called, service is paused");
    paused = true;
}

public void resume () {
    // things to do on service resume request

    logInfoEvent("resume() method called, service is resumed");
    paused = false;

    synchronized(pauseLock) {

        // unlocking service run thread
        pauseLock.notify();
    }
}
}

```

Note: The `run()` handler is long-running and thus is not limited in time. Other handler methods are required to complete within a particular time frame. The default timeout value is set to 10 seconds, however, `init()`, `stop()`, `pause()`, and `resume()` handlers can prolong the period of their execution by calling the respective `WinService` method, e.g. `setInitTimeout()`. The timeout for the `shutdown()` handler is defined by the system (as less than 20 seconds) and cannot be adjusted. If the method `init()` has not completed in time, the system logs a warning message into the system

event log and assumes that the service does not support pause/continue requests.

2.2 Building the service executable

To build a Windows service program using the JET Control Panel, do the following:

1. Compile the service main class:

```
javac -classpath JET-dir\lib\WinService.jar HelloWorld.java
```

where *JET-dir* is the JET installation directory, e.g. C:\JET.

2. Launch the JET Control Panel and create a new project. On the **Start** page choose “Windows Service” as the type of the file you want to create.¹
3. Add classes/jars of your application to the classpath. For the above example, add just the `HelloWorld.class` file.

Note: `WinService.jar` gets precompiled into the JET runtime, so there is no need to add it to the classpath.
4. Select *Service main class*, i.e. the class you created which extends `com.excelsior.service.WinService` (`HelloWorld` in the above example).
5. On the **Target** page, specify the executable name and NT service name. The NT service name is the system name that will be used to install, remove and manage the service. It may also be used to distinguish messages from the service written into the system event log.
6. On the **Compile** page, simply click the “Build” button, answering ‘OK’ in the appeared dialog.

2.3 Service installation and removal

For service testing purposes and for use with custom/third-party installation programs, Excelsior JET includes a command line utility called `isrv`, which allows you to install and remove your service to/from the system.

For example, to install the `HelloWorld` service you have just built, go to the build directory and issue the following command:

```
isrv -i HelloWorld.exe -displayname "HelloWorld Service"
```

The above command installs the service program `HelloWorld.exe` from the current directory and specifies that it has to appear in the list of services under the name "HelloWorld Service". Other `isrv` options have been left to defaults, which are as follows:

- The service will start automatically on system startup
- The service will run under the `LocalSystem` account
- The service cannot interact with desktop

To remove the previously installed service, issue the following command:

```
isrv -r HelloWorld.exe
```

For more `isrv` options and examples, please refer to Excelsior JET for Windows User’s Guide (Chapter “Windows NT Services”, Section “Service installation and removal”).

¹In Excelsior JET 4.0 and below you need to select “NT Service” as the desired target application type and “Manual settings” as the project type.

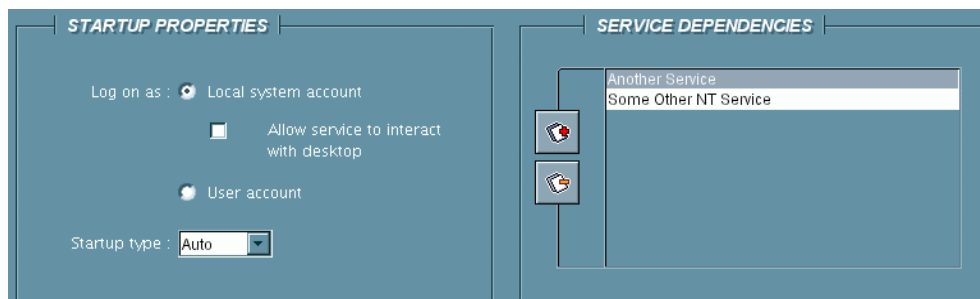
2.4 Deployment

To create an installation package for your service, run **JetPackII** from your Excelsior JET Start Menu and do the following:

1. Set up the project as usual.
2. On the **Backend** page, select **Excelsior installer**.
JetPackII will recognize that the project contains a service and enable the **Services** page.
3. On the **Services** page, you may enter a display name for your service. The display name is a name which will be shown in the system services list and in the Event Viewer system tool. Also it is possible to describe your service in the **Description** text box.



There are also several system options controlling the service's behavior in the system.



- (a) **Log on as** — specifies an account to be used by the service. There are two options:
- **Local system account** — run the service under the built-in system account.
 - **Allow service to interact with desktop.** If the local system account is used, a user may explicitly specify that the service may interact with the system desktop, i.e. open/close other windows, etc.
 - **User account** - run the service under a user account. When the package is deployed, the user is prompted for an account name and password.
- (b) **Startup Type** — specifies how to start the service. There are three types of startup:
- **Automatic** - specifies that the service should start automatically when the system starts.
 - **Manual** - specifies that a user or a dependent service can start the service. Services with Manual startup type do not start automatically when the system starts.
 - **Disabled** - prevents the service from being started by the system, a user, or any dependent service.

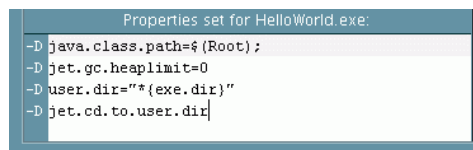
(c) **Dependencies** — the list of other services the service depends on.

4. Finally, go to **Finish** page, select the preferred package location and click **Create**. Now you can install the resulting package containing your service on other machines.

Hint: A service starts up with the current directory set to the Windows 32-bit system directory, regardless of where the service executable resides. If your service needs to know its installation directory (e.g. for reading/writing files), you have two options:

- Define the following properties on the **Resources** page:

```
-Duser.dir="*{exe.dir}"  
-Djet.cd.to.user.dir
```



This will result in JET runtime changing the current directory to the directory containing the service executable on executable startup.

- A simpler way would be to set your own property for the service executable, defined using the package root directory, for instance:

```
-Dmy.service.dir=${Root}
```

After installation with Excelsior Installer the value of `my.service.dir` property will be the full path to the root directory of the installation package.

REFERENCES

1. For further information, please consult Excelsior JET for Windows User's Guide available at <http://www.excelsior-usa.com/jetdoc.html>, Chapter "Windows NT Services".
2. Javadoc for Excelsior WinService API may be found in your Excelsior JET installation directory as `doc\index.html`.
3. Check out the NT services samples located under `samples\services` in your Excelsior JET installation directory.
4. Read more about Windows NT Services on [MSDN](#).