

WinCron .NET

Powering your servers since 1989

WinCron .NET Users Guide

Version 1.0

Wednesday, January 31, 2007

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

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

WinCron® is a Registered Trademark of Tomasello Software LLC.

Table of Contents

Table of Contents.....	2
Introduction	3
Overview of WinCron .NET	3
Installation.....	4
Getting started	4
The Basics	7
WinCron .NET Commands	7

Introduction

WinCron .NET is task scheduling software designed for the IT professional, systems specialist, or systems integrator. (WinCron .NET is not intended for the casual home user.)

WinCron .NET is a Windows system service with a telnet 'shell' frontend that uses standard CRON syntax for scheduling jobs.

WinCron .NET employs the full capabilities of the .NET 2.0 Framework Common Language Runtime (CLR) for maximum programmability.

Programmable in either Visual Basic or C#, WinCron .NET is unmatched in power and flexibility.

WinCron .NET hosts the CLR, so besides a text editor, no other tools are required.

If you've done any programming in perl, CGI, VB, etc. then you'll have no problem getting up and running with WinCron .NET.

Overview of WinCron .NET

If you've done any programming, then WinCron can actually be *easier* than most other scheduling software. The reason is simple: You don't have a User Interface standing between you and the task at hand.

If you simply want to run a program at 3PM, then most any scheduling software will work fine. If you want the running of that program to reoccur each weekday at 3PM, except Saturday and Sunday, *most* Task Schedulers will work for you. But if you want to say; open a text file, search it for the phrase "connection error", extract that line, and then have it emailed it to you, then you're probably going to need the power of WinCron .NET.

WinCron .NET is basically a telnet frontend to a system service based CLR with high level services for managing your jobs.

'Jobs' in WinCron .NET are simply VB .NET or C# scripts that are registered to run at a specific time.

Here is the quintessential "Hello World!" script.

```
/* Module: User.Hello
 *
 * Description:
 * Say "Hello World!" once per minute
 */

using System;
using WinCron;

namespace User
{
    class Hello
    {
        public static void Initialize()
        {
            Cron.Register("Hello", new Hello().OnCronEvent, "** * * * *");
        }
    }
}
```

```
public void OnCronEvent()  
{  
    Tools.PrintLine("Hello World!");  
}  
}
```

It may look like a lot of stuff for a simple "Hello World!" script, but this is really a complete frame work for any task, regardless of how complex it may be.

You can pretty much ignore all of the script except for the magic 3

1. Initialize()

Most WinCron .NET scripts will have an Initialize() function to register any tasks that need to be carried out at a certain time. The Initialize() function is called by WinCron .NET at load time. I.e., before anything else in your script runs.

2. Cron.Register()

The Cron.Register function registers your job—in this case 'Hello'—to run at a specific time. In this example we use the CRON syntax for every minute: "* * * * *"

Please see the section on "UNIX Cron Style Matching" for more information on forming a time specification.

3. OnCronEvent()

The OnCronEvent is the function that is called by the CRON system when the time expires. The OnCronEvent can do any required work and can take as much time as necessary to complete the work.¹

Installation

After downloading the setup package (setup.exe), you simply run it to install WinCron .NET. If you do not have the appropriate version of the .NET Framework, that too will be installed.

Getting started

After installing WinCron .NET, you connect to it with telnet.exe to view it's operation, start / stop jobs, list jobs, kill jobs, etc.

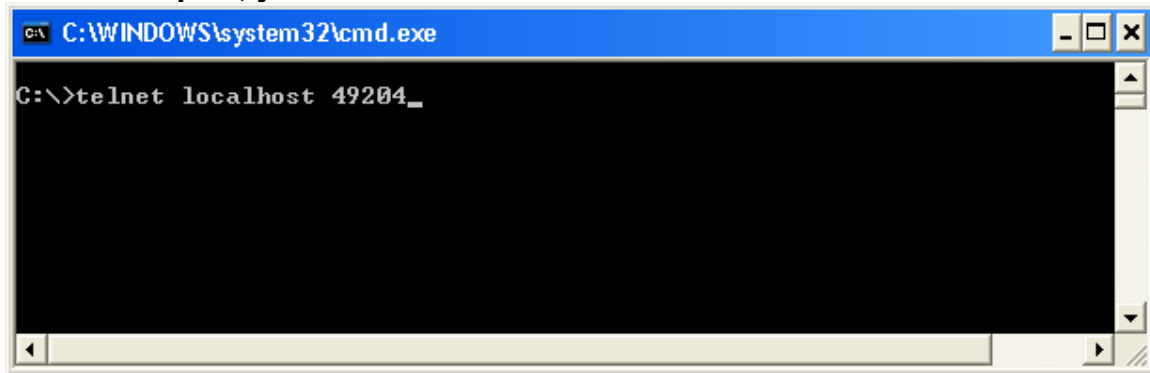
First, make sure WinCron .NET is running.

Control Panel/Administrative Tools/Services

If WinCron .NET is not running, right-click it and select 'Start'.

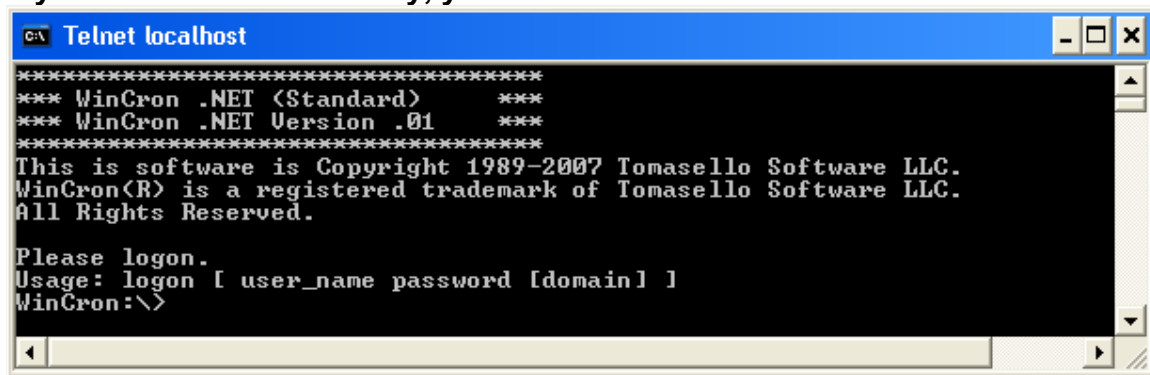
¹ WinCron .NET is multi-threaded, meaning it will continue to schedule tasks *on time* even while another task is still running. The scheduled tasks are kept in a queue and run in a serial mode as soon as possible.

Assuming WinCron .NET is running locally and that you have not changed the default port, you will connect to it like this:



```
C:\WINDOWS\system32\cmd.exe
C:\>telnet localhost 49204_
```

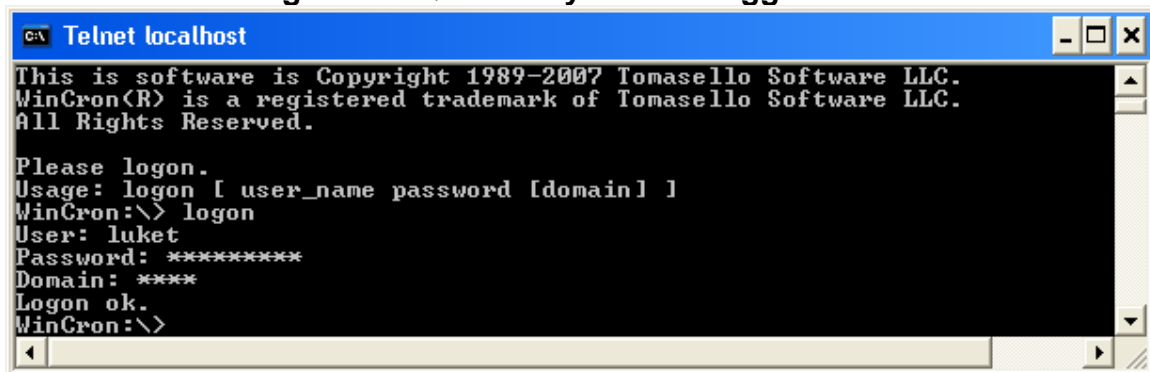
If you connect successfully, you will see a screen similar to this one:



```
Telnet localhost
*****
*** WinCron .NET (Standard) ***
*** WinCron .NET Version .01 ***
*****
This is software is Copyright 1989-2007 Tomasello Software LLC.
WinCron(R) is a registered trademark of Tomasello Software LLC.
All Rights Reserved.

Please logon.
Usage: logon [ user_name password [domain] ]
WinCron:\>
```

For security reasons, WinCron .NET requires that you have an account on the machine you are connecting to. WinCron will not respond to any commands but Logon and Quit until you have logged in.



```
Telnet localhost
This is software is Copyright 1989-2007 Tomasello Software LLC.
WinCron(R) is a registered trademark of Tomasello Software LLC.
All Rights Reserved.

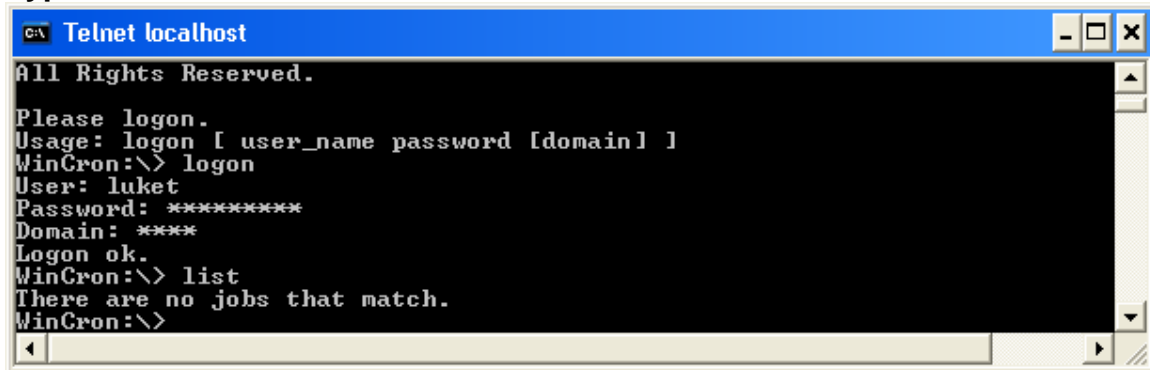
Please logon.
Usage: logon [ user_name password [domain] ]
WinCron:\> logon
User: luket
Password: *****
Domain: ****
Logon ok.
WinCron:\>
```

If you are part of a domain, you will need to enter that as well as your domain login info (user name and password.) If you are not part of a domain, or simply want to log into a local machine account, then just press <enter> when asked for the domain, or enter a single period '.' as that's shorthand for *this machine*.

You're all set to start having fun now! Woo!

We'll learn all about WinCron .NET Commands in an upcoming chapter, but for now, let's just try something really simple to get you started.

Type in *list*



```
C:\ Telnet localhost
All Rights Reserved.

Please logon.
Usage: logon [ user_name password [domain] ]
WinCron:\> logon
User: luket
Password: *****
Domain: ****
Logon ok.
WinCron:\> list
There are no jobs that match.
WinCron:\>
```

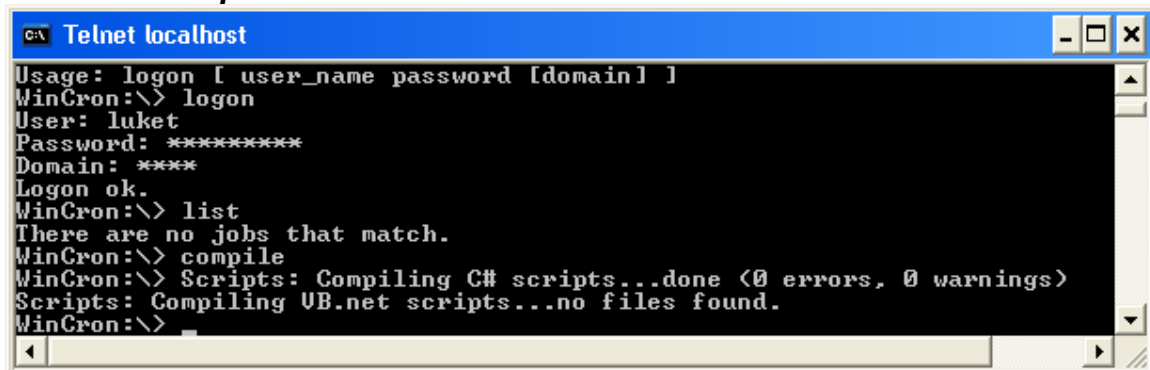
Now go to the folder where you installed WinCron .NET and copy the Hello.cs file from the Sample Scripts folder to the Scripts folder. If you installed WinCron .NET in the default location, that would be:

C:\Program Files\Tomasello Software\WinCron\Sample Scripts\Hello.cs
C:\Program Files\Tomasello Software\WinCron\Scripts\Hello.cs

 All scripts in the Scripts folder are compiled and loaded by WinCron .NET automatically on startup.

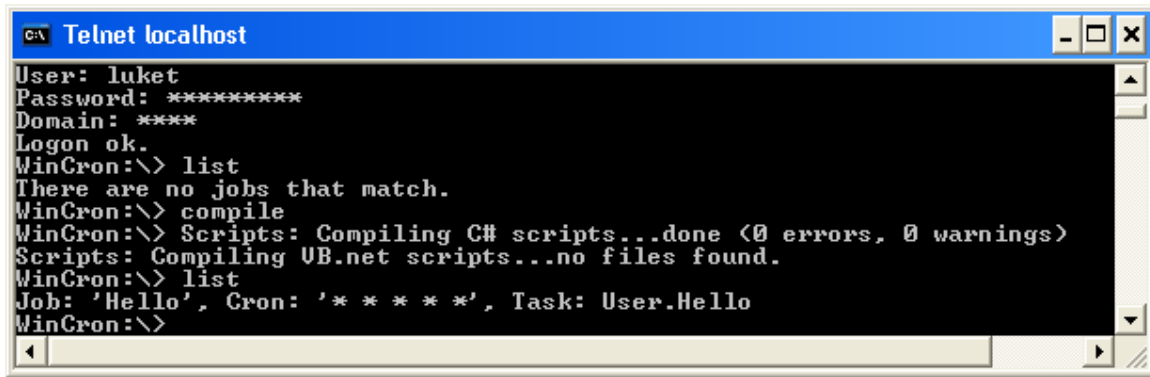
After the Hello.cs file is copied over, we need to tell WinCron .NET to recompile the Scripts folder.

Issue the *compile* command:



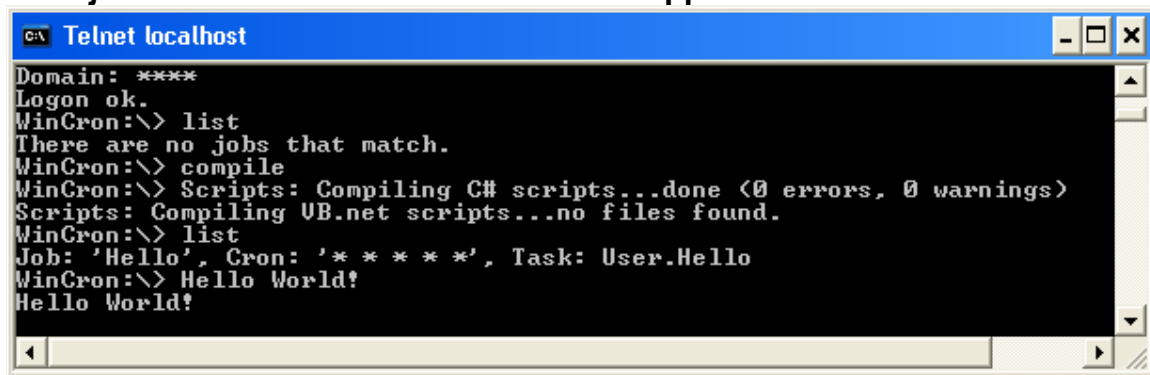
```
C:\ Telnet localhost
Usage: logon [ user_name password [domain] ]
WinCron:\> logon
User: luket
Password: *****
Domain: ****
Logon ok.
WinCron:\> list
There are no jobs that match.
WinCron:\> compile
WinCron:\> Scripts: Compiling C# scripts...done (0 errors, 0 warnings)
Scripts: Compiling VB.net scripts...no files found.
WinCron:\>
```

Now issue the *list* command again:

A screenshot of a Telnet window titled "Telnet localhost". The session shows a user logging in as "luket" and then interacting with the WinCron service. The user enters "list", "compile", and "list" commands. The "compile" command shows that C# scripts were compiled successfully with no errors or warnings, and that no VB.net scripts were found. The "list" command shows a job named "Hello" with a cron expression of "* * * * *" and a task of "User.Hello".

```
c:\ Telnet localhost
User: luket
Password: *****
Domain: *****
Logon ok.
WinCron:\> list
There are no jobs that match.
WinCron:\> compile
WinCron:\> Scripts: Compiling C# scripts...done (0 errors, 0 warnings)
Scripts: Compiling VB.net scripts...no files found.
WinCron:\> list
Job: 'Hello', Cron: '* * * * *', Task: User.Hello
WinCron:\>
```

As you can see our Hello.cs job compiled without error and loaded!
Let's just wait for a minute and see what happens...

A screenshot of a Telnet window titled "Telnet localhost". The session shows the same user logging in and interacting with the WinCron service. The user enters "list", "compile", and "list" commands. The "compile" command shows that C# scripts were compiled successfully with no errors or warnings, and that no VB.net scripts were found. The "list" command shows a job named "Hello" with a cron expression of "* * * * *" and a task of "User.Hello". The user then enters "Hello World!" and the output "Hello World!" is displayed.

```
c:\ Telnet localhost
Domain: *****
Logon ok.
WinCron:\> list
There are no jobs that match.
WinCron:\> compile
WinCron:\> Scripts: Compiling C# scripts...done (0 errors, 0 warnings)
Scripts: Compiling VB.net scripts...no files found.
WinCron:\> list
Job: 'Hello', Cron: '* * * * *', Task: User.Hello
WinCron:\> Hello World!
Hello World!
```

As you can see, each minute our Hello Job executes and prints "Hello World!" on the console.

That's the basic model for writing scripts; you create/modify scripts in the WinCron .NET Scripts folder, then tell WinCron .NET to recompile them. Once you are happy with your script(s), you can simply forget them as WinCron .NET will automatically load them each time the computer is started.

In the next section, we will learn about all the WinCron .NET commands; that is, the commands that you can type in from the telnet session like *list* and *compile* above.

WinCron .NET Commands

In the last section, we connected a telnet session to WinCron .NET to run a simple Job. In this section, we will list the commands that are available from within the telnet session:

- * **help [*command*]**
Lists help for all commands or a specific command.
- * **logon [*user_name password [domain]*]**
Logs onto the computer hosting the WinCron .NET service

You must have an account on the computer you are trying to connect to.

For 'over the shoulder security', you can simply type *logon <return>* and WinCron .NET will prompt you for the information and mask your password as it is typed. I.e., "*****"

* **license**

The license command displays the WinCron .NET startup banner which shows the current license state of the software: *Evaluation, Standard, Professional, and Enterprise.*

* **quit**

Quits the current telnet session and disconnects from the WinCron .NET server. The server continues to run and execute scripts.

* **kill jobs**

Kills the job or jobs specified by the *jobs* parameter.

The *jobs* parameter is a regular expression so multiple jobs can be matched.

Example: kill log.*

The above example would kill the matching jobs *LogRotate, LogDelete, and LogReport.*

* **echo**

Sends a message to all connected telnet clients.

* **compile**

Compiles and loads (and begins execution) of all scripts in the Scripts folder.

There is no need to stop and restart the WinCron Service when compiling jobs. This greatly speeds script debugging and development.

* **list [jobs]**

Lists the job or jobs specified by the *jobs* parameter.

The *jobs* parameter is a regular expression so multiple jobs can be matched.

Example: list log.*

The above example would list the matching jobs *LogRotate, LogDelete, and LogReport.*

* **compact**

Requests the WinCron .NET server—and more precisely the Microsoft CLR—to perform garbage collection.

Note: This is a debugging aid and rarely used.

* **register "your name" registration-code**

This command registers the product in your name (or your companies name). You obtain a registration code by purchasing a license key from [RegNow](http://www.regnow.com).

<https://www.regnow.com/softsell/nph-softsell.cgi?item=1754-2>

* **run job**

You may at times want to run jobs immediately, circumventing their normal scheduled execution.

After issuing the run command, jobs will continue to execute on schedule.

* **diag**

Used for debugging, the diag command dumps strings printed to a special diagnostic buffer.

You use the built-in output method in your scripts:

```
WinCron.Diag.WriteLine("some string {0}", some-variable);
```

to print interesting or other diagnostic information.

The data printed with the `WinCron.Diag.WriteLine()` method is not displayed on the console until the diag command is issued; this keeps the normal display output clean.

The Basics

Miscellaneous Configuration

In the WinCron.ini file in the WinCron .NET installation folder contains at least the following configuration settings.

```
; Server settings
[Server]
; Choose a port from the 49152-65534 range.  Default: 49204
Port=NNN

[Data]
; User data variables.
; you can name them anything you like, but I would recommend a
convention
; something like: class.method.variable
; This convention allows you to easily map variables in this file with
the
; actual variables in the script. See the scripts GameServer.cs and
WebServer.cs
; for an example of this.
; Use the method Tools.ReadVariable("var-name") to read this data
MyData.Something=Here is some sample runtime data!
```

This file provides a convenient place to store server, task, or user specific data that doesn't otherwise belong in a script. A good example of the types of data that you would want to store here would be things like user names, passwords, SMTP server names, etc.

You can write generic scripts, and put the 'task specific' data here in WinCron.ini.

UNIX Cron Style Matching

The general format of a UNIX cron-style pattern is:

"minute hour day-of-month month day-of-week"

WinCron .NET's UNIX cron-style matcher is based on Paul Vixie's work which is upward compatible with the V7 standard.

The WinCron .NET pattern matching engine recognizes the following fields:

Field	Meaning	allowed values
1	Minute	0-59
2	Hour	0-23
3	day of month	1-31
4	Month	1-12
5	day of week	1-7 (1=Sun, etc.)

A field may be an asterisk (*), which always stands for "first-last". Ranges of numbers are allowed. Ranges are two numbers separated with a hyphen. The specified range is inclusive. For example, 8-11 for an "hours" entry specifies execution at hours 8, 9, 10 and 11.

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

Step values can be used in conjunction with ranges. Following a range with `/` specifies skips of the number's value through the range. For example, `0-23/2` can be used in the hours field to specify command execution every other hour (the alternative in the V7 standard is `0,2,4,6,8,10,12,14,16,18,20,22'). Steps are also permitted after an asterisk, so if you want to say `every two hours', just use `*/2'.

Examples:

7:00am each weekday [mon-fri]	00 07 * * 2-6
1st of each month, at 5:30pm	30 17 1 * *
at 8:00am,10:00am and 2:00pm every day	00 8,10,14 * * *
every 5 minutes during market hours [mon-fri]	*/5 6-13 * * 2-6
every 3-hours while awake	0 7-23/3 * * *
annually Run once a year	0 0 1 1 *
monthly Run once a month	0 0 1 * *
weekly Run once a week	0 0 * * 0
daily Run once a day	0 0 * * *
Midnight (same as daily)	0 0 * * *
hourly Run once an	0 * * * *

hour	
------	--

Registering WinCron .NET

WinCron .NET is delivered in an 'evaluation' form, and except for a 3 job limitation, it is full featured.

After determining that WinCron .NET will satisfy your needs, you purchase a license 'key' to unlock the job count limitations, and to remove the 'registration reminders'.

You obtain a registration code by purchasing a license key from [RegNow](https://www.regnow.com/softsell/nph-softsell.cgi?item=1754-2).
<https://www.regnow.com/softsell/nph-softsell.cgi?item=1754-2>