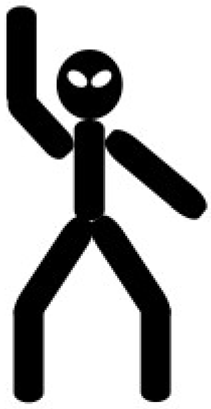
CHAPTER 1

Hello NetKernel



Welcome. If you are reading this you've probably heard **NetKernel™** mentioned in some XML-related workshop, pretty much like I did back in 2006. Or maybe you have heard about **Resource Oriented Computing** (ROC)**™** and want to see a practical implementation. Or maybe ... Whatever your reasons for reading it, this book intends to take your hand show you the wonderful world of NetKernel[1]. Are you ready ?

Audience

This book is intented for beginning and intermediate ROC-ers[2]. There is a learning curve to ROC and NetKernel and this book will help you along that curve.

Conventions

The book was made with OpenOffice 3.2.1, all formats/fonts mentioned below are available in that editor.

The standard font for this book is **Verdana, 12pt, Black**
Chapter titles are **18pt, Turquoise 6**
Subtitels are **14pt**, **Turquoise 5**
Headings are **12pt, Bold, Black**

Operators (verbs indicating an action) in the text are underlined, followed by an operand (the thing acted upon) in italic.

<u>Push</u> *this*
<u>Shake</u> *that*

The font for operating system output in this book is **Courier New, 12pt**, instruction lines have a **Grey (10%)** background, the instructions themselves are put in **Bold**.

```
your_user@ubuntumachine:~$ aptitude -vvvv moo
Okay, okay, if I give you an Easter Egg, will you go away?

your_user@ubuntumachine:~$ aptitude -vvvvv moo
All right, you win.
```

1 1060, NetKernel, Resource Oriented Computing, ROC are respectively registered trademark and trademarks of 1060 Research Limited
2 Puns with *ROC(ks)* are encouraged in all use of NetKernel. It does make asking for a drink with ice at a NetKernel convention a rather tricky thing to do though ...

```
                               /----\
                       -------/      \
                      /               \
                     /                |
   -----------------/                  ---------\
   ----------------------------------------------
```

```
your_user@ubuntumachine:~$ aptitude -vvvvvv moo
What is it?  It's an elephant being eaten by a snake, of
course.
```

Content

3 I did warn you about the puns !
4 http://www.mongodb.org/

Content (continued)

Don't know much about history ...

1060 Research was founded as a spin-out of original research (codenamed *dexter*) undertaken at Hewlett-Packard labs. The 1060 team created and implemented the **Resource Oriented Computing™** model in what you and I know as **NetKernel™**. In the mean time this technology is more than 10 years old and has proven itself in sectors ranging from Telecoms, Insurance, Banking and Military.

At the time of writing this book (September 2010) my source at Hewlett-Packard labs tells me that the current research – something that can be used to automate your garden management (amongst other things) – is codenamed *dharma*[5]. Now, aren't you glad you just found out that bit of information ? Personally I just wonder if they work alphabetically.

If you haven't lost your sense of humour by now, how is your knowledge of Roman numerals[6] ? Perfect ? Then try this :
X + M + L = ?

10000 feet view of Resource Oriented Computing

If you want to a more complete (and correct) explanation than you'll get here, have a look at ***Introduction to Resource-Oriented Computing, part I***, an excellent document that you can find under the heading *Technical Whitepapers* on http://www.1060research.com/netkernel/roc/. Be warned though that it is a tough read and that intimate knowledge of both *Plato* and *Jack and the beanstalk* are required.

Let me tell you what I think ROC is (after working on and off with it for +/- 4 years) :

1. Everything is a resource. That includes your code, my code, data on the database on my workstation and data encrypted steganographically in an image of the National Art Gallery in Kuala Lumpur.
2. Once you grasped 1. (and that may take a while) let go of worrying about where and how those resources are implemented.

5 http://en.wikipedia.org/wiki/Dharma
6 http://en.wikipedia.org/wiki/Roman_numerals

3. Instead, focus on what you want to do with the resources, doing that in small simple services that you can string together to as complex a system as you can imagine (and probably way beyond that).
4. Stand amazed at how your system scales in exactly the same way that the internet scales.

Why bother ?

When I was in school (*Anno Domini Nostri Iesu*[7] 1992) training to become an IT Bachelor, we got an introduction session on the NeXTSTEP platform. The platform that was going to make us (IT professionals) obsolete within the next couple of years. And after the impressive session most of us (including myself) believed it.

18 years later I'm still in IT. People still use Cobol, PL1 and CICS (as they – well, obviously not the same people ... I hope – did before I was born). NeXTSTEP only survives in the Apple OS somewhere.

If you've been around in IT for a while, you'll have such a story of your own. There's always the **next** best thing that will take away all the pain of software development (the human factor mostly) completely. And managers will always love it.

But if one really has to say what technology **has worked** in the last 15 years, one would have to say ... the internet. It has grown beyond imagination.

ROC™ combines the core ideas of the internet, the core ideas of Unix[8] and REST[9] into a new and potent whole :

> *From Unix* - borrow the idea of using simple tools that share a common interopable data model (e.g., awk, grep, sed, etc.) to build solutions.
>
> *From REST* - address everything (resources, services and code) with a URI to loosely couple the *internals* of your software making it as flexible as the Web.

NetKernel™ brings all this to an infrastructure near you !

Don't take my word for it. In retrospect I'd have liked to hear the guy showing us NeXTSTEP say that. In fact, our class did meet him again as a mime (doing a robot-impersonation) on some IT gathering later that same year. It turned out that he was out of a job only weeks after giving us the presentation.

As of Chapter 2 each chapter will contain a bit of ROC-talk (indicated by the 'Rolling Stones' image) and a lot of hands-on NetKernel-stuff you **can** try at home. In fact, I'm counting on you to try it at home !

7 In
8 http://en.wikipedia.org/wiki/Unix
9 http://en.wikipedia.org/wiki/Representational_State_Transfer

CHAPTER 2

Stacking the deck

Prerequisites

NetKernel
Installed and running. Appendix A explains how to accomplish that.

A brain[10]
These come in all shapes and sizes. Mine – for example – is not special in any way. Brain-flexibility is required though. ROC is not *difficult*, just *different*. Remember that – contrary to common belief – new braincells can be added and new pathways through your brain can be created. And after you've worked with ROC for a while both those statements will become fact !

A texteditor
There are many good plaintext-editors. And if you've written any code at all you'll probably have a favorite one. I personally use SciTE because it is lightweight, portable and very customisable. In the documentation that comes with your NetKernel instance (I'll show you where you can find that documentation a bit further on), IntelliJ IDEA is suggested.

Stick with whatever makes you productive. If your editor has code-highlighting for the most common stuff (xml, html, css, java, javascript, ...), you'll be fine. Only if the default Windows notepad is your top-of-the-line texteditor I would strongly suggest you to follow the above suggestion[11].

10 My brain related knowledge comes from http://pragprog.com/titles/ahptl/pragmatic-thinking-and-learning, an excellent (very readable, even for the technically inclined) book on the matter.
11 No, I will not start a flame war over this. If you are happy with notepad, kudos to you !

Whenever you see the 'Rolling Rocks'-sign in the book we are going to have a bit of ROC-talk. Don't worry, I'll try to keep it clear and short at all times. Every sign will also be a link to the next ROC-talk, so if you want to walk through all of them at once, you can (in an electronic version of this book that is) ...

ROC has its own principles and terminology. This ROC-talk will cover the first three (of seven) principles :

1. *A resource is an abstract set of information*
   Example : There is a book called 'Hello NetKernel'.

2. *Each resource may be identified by one or more logical identifiers*
   Example :
   - The book 'Hello NetKernel' is the book refered to here.
   - The book 'Hello NetKernel' is the book described here.
   - The book 'Hello NetKernel' is the book I am writing right now.

3. *A logical identifier may be resolved within an information context to obtain a physical resource-representation*
   Example :
   - As the odt-file I am editing right now.
   - As the pdf-file you can read.
   The verbs (editing and read) form the **information context**, the hyperlinks are **logical identifiers**, what you get on screen (even if you get an error, as you probably will in the case of the file, since that is locally on my machine) is the **physical resource-representation**.

Now, that was not difficult at all, was it ? And we got some terminology done as well, excellent. Do not worry if you can not *place* this information yet, you soon will.

Setup

We are going to dive straight in. Let us first have a look at what we've got and explain a couple of things along the way.

Layout NK4 installation

In what follows I'm labeling the *location* of your NK4 installation as **[install]**. I'm also going to use forward slash to indicate a directory. I know this is different in Windows[12], you'll soon notice however that within NetKernel configuration files (regardless of the operating system) the forward slash is used and I did not want a mix. So adjust for Windows where necessary. I'm also going to call your running NetKernel the **[instance]**.

When you look into your [install] directory you'll see these subdirectories :

bin	startup script and startup configuration files
etc	[instance]-wide configuration files
javadoc	generated documentation
lib	[instance]-wide libraries
log	loggings
modules[13]	the NetKernel batteries (applications and tools)

There are a couple of others. These are volatile directories for caching and for the H2[14] databasefiles used by the [instance] itself.

[install]/bin

Only the scripts to manually start a NetKernel instance and the configuration files containing the parameters for those scripts can be found here.

[install]/etc

There's some interesting stuff here. The *kernel.properties* file contains the parameters that govern your NetKernel instance. Very interesting stuff, but do not touch unless you have a very good reason (and know what you are doing). Besides, you can change all of these parameters from the Backend GUI.

The *modules.xml* file contains which modules (applications, tools) get loaded (and in which order). You will modify this file. Either manually or through the Backend GUI, but this is where you will add your own modules. In case it was not clear yet, you'll now – by looking at modules.xml – realize that NetKernel is build up from modules that run ... in NetKernel.

12 A functional guy in my company put in a request to our Windows System Administration team to adjust all backward slashes to forward slashes. They forwarded the issue to Microsoft Support. No answer has come from Redmond so far.
13 Module is the generic name for application or tool within NetKernel
14 http://www.h2database.com

[install]/javadoc
Statements :
1. NetKernel is developed in Java.
2. Java is *one of the languages* you can use to develop modules in NetKernel.

Right, that's out of the way ! I'm by no means a Java-guru (I prefer Python, sorry). The NetKernel developers used Java, the modules that make up NetKernel are Java modules. These modules can generate javadocs that put their documentation in the javadoc-directory. If you develop your own modules for NetKernel in Java, so can they.

People are always giving me a hard time with the "What is NetKernel ?"-question. "Is it an application-server ?", they ask. "Well, no, not really", I answer. "O, then it is an alternative for Java.", they retort and end the discussion ... (which should actually just start then). For me this just proofs some people should stay as far away from IT as possible.

[install]/lib
This directory contains the libraries used to boot the NetKernel instance itself. Note that when I state *[instance]-wide* I mean you'll have a similar directory for each of your own applications as well (*application-wide* in other words). The libraries in *[install]/lib* provide the actual ROC-functionality.

[install]/log
Guess what, this directory contains the loggings of your NetKernel instance. No need to study them here, the Backend GUI contains a very nice logviewer.

[install]/modules
It is very hip to say something is *batteries included*. Python seems to be. Haskell too. I guess Ruby couldn't stay behind. But what does it mean ? What does it mean to say *NetKernel is batteries included* ?

Well, while having the ROC-functionality at your fingertips is surely very nice, it means absolutely nothing to me. I need documentation about it, I need to be able to see it, I need to be able to use it. Those are the batteries ! And that is what the *[install]/modules* directory contains, applications and tools that use the ROC-functionality and open it up to a simple soul like me.

So, if I want to use Python to ROC in NetKernel, I can. Some Saxon for XML processing ? No problem ! Ant ? Sure ! Those and many other things are available ... Some are installed by default, others can be gotten from the repositories.

[install]/project-modules
Is not in the list. Create *it* now. This is where you'll put your own applications. Using *project-modules* as directory-name is not mandatory (you can choose whatever you like but please do not put blanks in it), I'll stick with *project-modules* for the rest of this book.

```
cd [install]
mkdir project-modules
```

Glossed over

You – readers of this book – are not stupid. You no doubt noticed that I glossed over a couple of things :

1. I did not give my answer to the "What is NetKernel ?"-question.
2. I did not explain what ROC-functionality means.
3. I keep refering to wonderful stuff in the Backend GUI, but I do not show any of it.
4. ...

The reason is that I want you to be able **to do something** (almost there) before I get to page 100 or so. As for the answer to 1., one of the purposes of this book is that you're able to formulate an answer to that yourself. So bear with me, all will be explained.

Hammer Time – your first module

Directory
We are going to create a directory for your module underneath *[install]/project-modules*. You could name this directory anything you like. I'm going to follow the URN[15] method used in the *[install]/modules* directory. Note that for a directory or file, you can not use colons. We use points instead. So these could be possible directory-names for the application :

```
urn.com.colruyt.tutorial.firstmodule-1.0.0
urn.org.tomgeudens.tutorial.firstmodule-1.0.0
```

I also add a version number in the directory name. This too is optional, just makes it easier if I want to have two versions of the same module running side by side.

15 http://en.wikipedia.org/wiki/Uniform_Resource_Name

My employer would love for you all to use the first option whereas I would of course love to become immortal by having a directory with my name in it on your harddrive. But I'll stay modest and go with this directory name :

```
urn.org.netkernelbook.tutorial.firstmodule-1.0.0
```

Create *the directory* for your module in *[install]/project-modules*.

```
cd [install]/project-modules
mkdir urn.org.netkernelbook.tutorial.firstmodule-1.0.0
```

I'm not differentiating between Windows and Linux when stating the commands, remember to use the non-superuser **dexter** though to execute the commands on Linux.

Module definition

Every module has (must have) a *module.xml* file in the root of its directory. Create *[install]/project-modules/urn.org.netkernelbook.tutorial.firstmodule-1.0.0/module.xml* (with the texteditor of your choice).

And here's the content (don't worry about what it all means right now, we'll go into exhaustive detail later on) :

```
<module version="2.0">

  <meta>
    <identity>
      <uri>urn:org:netkernelbook:tutorial:firstmodule</uri>
      <version>1.0.0</version>
    </identity>
    <info>
      <name>firstmodule</name>
      <description>Tutorial to create a first module</description>
    </info>
  </meta>

  <system>
    <dynamic />
  </system>

  <rootspace>

    <fileset>
      <regex>res:/etc/system/SimpleDynamicImportHook.xml</regex>
    </fileset>

    <mapper>

      <config>

        <endpoint>
          <grammar>res:/netkernelbook/firstmodule/hello</grammar>
          <request>
            <identifier>active:dpml</identifier>
            <argument name="operator">
              res:/resources/dpml/hello.dpml
            </argument>
          </request>
        </endpoint>

      </config>

      <space>
        <fileset>
          <private />
          <regex>res:/resources/.*</regex>
        </fileset>
        <fileset>
          <private />
          <regex>res:/etc/.*</regex>
        </fileset>
        <import>
          <private />
          <uri>urn:org:netkernel:lang:dpml</uri>
        </import>
        <import>
          <private />
          <uri>urn:org:netkernel:ext:layer1</uri>
        </import>
      </space>

    </mapper>

  </rootspace>

</module>
```

Dynamic Import
Again something we'll discuss in detail later, it basically means that we are making our module accessible from *outside* (outside NetKernel itself that is, so we can access it in our webbrowser).

Create directory *[install]/project-modules/urn.org.netkernelbook.tutorial.firstmodule-1.0.0/etc*

Remember that I described [install]/etc as the [instance]-wide directory for configurations ? Typically every module has its own etc directory with module-wide configurations as well.

Create directory *[install]/project-modules/urn.org.netkernelbook.tutorial.firstmodule-1.0.0/etc/system*

Create *[install]/project-modules/urn.org.netkernelbook.tutorial.firstmodule-1.0.0/etc/system/SimpleDynamicImportHook.xml*

And here's the content:

```
<connection>
  <type>HTTPFulcrum</type>
</connection>
```

The hello endpoint
This is the actual program, written in DPML, NetKernel's own scripting language. Whenever I can use DPML I will do so in this book. That way I avoid discussion over which language is best as well as leveling the playingfield.

Create directory *[install]/project-modules/urn.org.netkernelbook.tutorial.firstmodule-1.0.0/resources*

Create directory *[install]/project-modules/urn.org.netkernelbook.tutorial.firstmodule-1.0.0/resources/dpml*

Create directory *[install]/project-modules/urn.org.netkernelbook.tutorial.firstmodule-1.0.0/resources/html*

Create *[install]/project-modules/urn.org.netkernelbook.tutorial.firstmodule-1.0.0/resources/dpml/hello.dpml*

And here's the content :

```
<sequence >
  <request assignment="response">
    <identifier>res:/resources/html/hello.html</identifier>
  </request>
</sequence>
```

As you can see, the program uses another resource, which we haven't got yet, so create *[install]/project-modules/urn.org.netkernelbook.tutorial.firstmodule-1.0.0/resources/html/hello.html*

And here's the content :

```
<html>
  <head>
   <title>NetKernelbook First Module - Hello</title>
  </head>
  <body>
    <h1>NetKernelbook First Module - Hello</h1>
    <p>Your module has been sucessfuly generated and deployed.</p>
  </body>
</html>
```

Registering the module
NetKernel has to be made aware of our new module. The place to do that is in *[install]/etc/modules.xml*

Add *the following entry* just before the *</modules>* endtag in *[install]/etc/modules.xml* (entry is on one line, the split below is due to the limited linesize in this book) :

```
<module runlevel="7">project-
modules/urn.org.netkernelbook.tutorial.firstmodule-1.0.0/</module>
```

Testing
NetKernel should have discovered your module now. To make sure it did :
- fire up your favorite webbrowser
- enter http://localhost:8080/netkernelbook/firstmodule/hello

Source Version Control
It does seem ridiculous to bring this up here, but this is a good point to set up source version control. Every module you write – even one as small as this one – should have it. If you are not familiar with it, Appendix E will guide you. If you are ... use it.

Well done
You are probably not very impressed yet with the results. Rome was not build in one chapter either and trust me, we've covered a lot of ground already.

Conclusion

Chapter 2 was aimed at you getting your first module up and running as quickly as possible. Chapter 3 will take the scalpel to the same module and explain it in both ROC-terms and technical-terms.

CHAPTER 3

Incision, right here

Prerequisites

Chapter 2 completed.

Inside module.xml

If you've been awake during the previous chapter, you noticed that the *module.xml* file contains most of the meat.

It does in fact have two – main – purposes :

1. Define your module
2. Define the ROC layout of your module

One is obvious, two is important. If your ROC layout is done well, putting the rest of your module together will be easy (and short). So lets study it in detail.

Template

A simple template for *module.xml* might look like this :

```
<module version="2.0">

  <meta>
    <identity>
      <uri>urn:org:yourcompany:yourapplication:newmodule</uri>
      <version>0.0.1</version>
    </identity>
    <info>
      <name>New Module</name>
      <description>New Module Template</description>
    </info>
  </meta>

  <system>
    <dynamic />
  </system>

  <rootspace>
  </rootspace>

</module>
```

Having good *templates* available can seriously reduce your development time. You do not need a fancy system to manage them (some editors have one and if you feel comfortable with it, by all means use it), a simple directory will do. Make sure to put them under version control as well though !

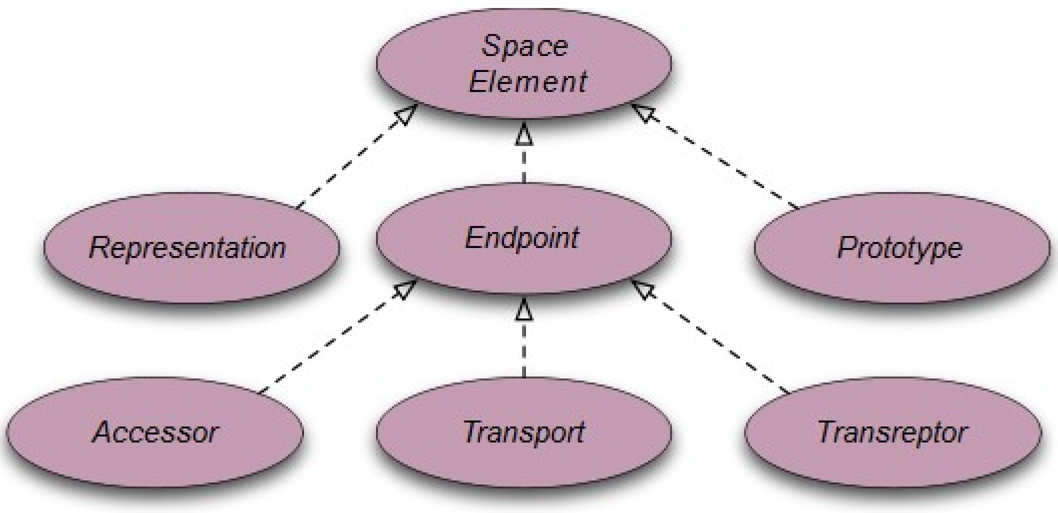
The template contains nothing fancy.

- meta
  Contains the description of your module.
  Note the use of colons in the *uri*.
- system
  With the dynamic-tag we indicate that any java-classfiles – within the application - that change during the run of the NetKernel instance will be unloaded and reloaded *dynamically* ...
  Note that this is not necessary for the other supported languages since those are not compiled and/or pre-loaded into the jvm and hence any change will immediately be active.
- rootspace
  Empty right now, this is where most of the action will take place.
  Note that you can have multiple rootspaces in one *module.xml*. If you do each will require its own unique uri :
  ```
  <rootspace
  uri="urn:org:yourcompany:yourapplication:newmodule:part1">
  </rootspace>
  <rootspace
  uri="urn:org:yourcompany:yourapplication:newmodule:part2">
  </rootspace>
  ```
  Any other modules that use (part of) your module, will have to import (we'll see that in a minute) the uri(s) of the relevant rootspace(s) rather than the uri of the module.

Template filled for firstmodule

uri	urn:org:netkernelbook:tutorial:firstmodule
version	1.0.0
name	firstmodule
description	Tutorial to create a first module

Rootspace
Within a rootspace we define a *resource space*.



Do not worry if the above diagram means very little yet. I'll explain everything as we go along. Ignoring representations and prototypes for now, an **endpoint** *is a space element that provides a gateway between logical resources and physical code*. That makes sense yes ? This will make it even clearer ... from the perspective of an endpoint, it participates in the two phases of request processing:

- **Request Resolution** - an endpoint may be asked to resolve a request.
- **Request Evaluation** - a resolved endpoint is asked to create and return a resource representation.

And to finish up defining a rootspace ... *a rootspace is a top-level space that can be imported into other spaces*.

So, in conclusion, your module has to contain a *rootspace* (one or more). In this rootspace you have the *endpoints* that define *what* your module will handle and *how* it will handle it. If you want to use the endpoints of other modules, you have to import the relevant rootspace. Vice versa, if other modules want to use your endpoints, they have to import your rootspace.

Let this information slowly sink in your mind. If it doesn't make sense yet, follow through the chapter, then come back and read the above again.

Rootspace definition

NetKernel is very flexible and can be adapted to suit your needs. This flexibility does have a downside for unexperienced users. There is – for example – *no single correct way* to define your resource space (endpoints). It depends.

In order to address this issue, enter **overlays**. According to the definition, *an overlay is an endpoint that defines and manages a relationship between a host space and a wrapped space*.

If that made sense to you, congratulations, you may move on to Chapter 4 ! The rest of the readers can follow me as we'll step-by-step reproduce the mapper definition of *firstmodule*.

Mapper template

```
<mapper>
  <config>
    <endpoint>
      <grammar />
      <request />
    </endpoint>
  </config>
  <space />
</mapper>
```

Needs some fleshing out, but otherwise complete. Let us start at the bottom with the space-tag.

Mapper template space-tag - fileset

This is what you'll find in *firstmodule*

```
<mapper>
  <space>
    <fileset>
      <private />
      <regex>res:/resources/.*</regex>
    </fileset>
    <fileset>
      <private />
      <regex>res:/etc/.*</regex>
    </fileset>
  </space>
</mapper>
```

The definition for **fileset** will make this clear ... *exposes physical resources contained within a module's physical directory to the logical space. All exposed files will be resolved using the generic* ***res:/*** *URI scheme.*
So, the *resources* and the *etc* directory within the module are accessible. For whom ? Well, the private-tag indicates that they are only accessible by the module itself.

Example – fileset
If at a certain point our module does the following resource request - res:/resources/dpml/hello.dpml - then by defining the above filesets the module knows that it can find the resource (a physical file in this case) hello.dpml in the resources/dpml directory of the module itself.

Mapper template space-tag - import
This is what you'll find in *firstmodule*

```
<mapper>
  <space>
    <import>
      <private />
      <uri>urn:org:netkernel:lang:dpml</uri>
    </import>
    <import>
      <private />
      <uri>urn:org:netkernel:ext:layer1</uri>
    </import>
  </space>
</mapper>
```

Remember that we stated that a rootspace can be imported into another module's space. Well, here we do this with
urn:org:netkernel:lang:dpml
urn:org:netkernel:ext:layer1

The rootspaces with those names are imported into our module. That means – amongst other things – that we get access to the DPML-interpreter and can write *active:dpml* requests. Yes, we get access to an execution environment, remember, **everything is a resource**, and that includes physical files as well as execution environments !

You might wonder what the *layer1* is about. Well, that module provides a lot of basic services in NetKernel (as well as *active:java*) and you'll see it imported in practically every module.

The private-tag indicates once more that any imported functionalities are only available for your module itself.

Mapper template space-tag – conclusion
With the space-tag we (as must be clear by now) defined where our module is going to find its resources. In the big-words definition for an *overlay* we called that the *wrapped space*.

Mapper template config-tag
In the config-tag we are going to define what our module is going to handle and how. That is the *host space*.

Mapper template endpoint-tag
Within the config-tag we have one or more endpoints. Here's the one for *firstmodule*
```
<endpoint>
  <grammar>res:/netkernelbook/firstmodule/hello</grammar>
  <request>
    <identifier>active:dpml</identifier>
    <argument name="operator">
    res:/resources/dpml/hello.dpml
    </argument>
  </request>
</endpoint>
```

As you can deduce, in a **grammar**-tag you define what is handled by the endpoint. In this case it handles the exact request for resource res:/netkernelbook/firstmodule/hello. That doesn't leave room for a lot. You can make things a lot more interesting of course, which we'll do a bit further down in this chapter.

The **request**-tag also speaks for itself. The above resource is going to be resolved by doing an *active:dpml* (present in our wrapped space !) with the operator-argument (= dpml program to be executed) coming from the resource request for res:/resources/dpml/hello.dpml (present in our wrapped space !).

Everything is a resource and all (!) NetKernel does is handle resource requests. In *firstmodule* these don't go very deep, but I hope you still get a *feel* of how deep this can go (as deep as you can imagine and beyond).

I skipped something
I've gone through most of *module.xml* by now, but I did skip one thing. The first thing in the rootspace of *firstmodule* is this
```
<fileset>
  <regex>res:/etc/system/SimpleDynamicImportHook.xml</regex>
</fileset>
```

And the contents of that local file are these
```
<connection>
  <type>HTTPFulcrum</type>
</connection>
```

Remember that we imported other rootspaces into our module ? And that it is possible that other modules import ours ? Well, some modules can do the latter thing (importing ours) automatically. That is called *dynamic importing*.

By exposing (see below) the *SimpleDynamicImportHook.xml* file, we tell those modules that we are ready to be imported. Which modules ? Well, the ones mentioned in the file (in this case HTTPFulcrum). Why ? Well, I doubt that if you enter res:/netkernelbook/firstmodule/hello in your browser, your going to have much success. You are going to enter something like http://localhost:8080/netkernelbook/firstmodule/hello and on port 8080 the HTTPFulcrum is listening and it will look at your http-request and see if there is a module that can handle it. There is ... *firstmodule* can handle it, but only if HTTPFulcrum knows about it.

A word on exposing
When I started explaining about the rootspace definition, I said that there's a lot of ways to do so. And there is, actually, our *firstmodule* rootspace is not only exposing res:/netkernelbook/firstmodule/hello, but also res:/etc/system/SimpleDynamicImportHook.xml, by using a simple fileset-tag.

All right, that covers the *firstmodule*'s *module.xml* file. Next on the menu is a ROC-talk and we'll finish up this chapter by expanding *firstmodule* a bit and by using NetKernel as a webserver.

If you've followed (and understood) this chapter so far, the ROC definitions should start to make sense. In this ROC-talk we'll cover the remaining four principles.

4. *Computation is the reification*[16] *of a resource to a physical resource-representation*
   Example :
   - Opening this odt-file in Open Office
   - Requesting this pdf-file in a webbrowser
   The verbs (opening, requesting) are the **computation**, the hyperlinks are **logical identifiers**, what you get on screen (even if you get an error, as you probably will in the case of the file, since that is locally on my machine) is the **physical resource-representation**.

5. *Resource-respresentations are immutable*
   Example :
   - I'd be very surprised if I opened the above file and found an explanation of how to grow peas. So would you if you requested the above url.
   - As I update this book you will time to time get a newer version (= another resource-representation) of the book when you request the same url.

6. *Transreption is the isomorphic*[17] lossless transformation of one physical resource-representation to another
   Example :
   - D:\In Progress\hello_netkernel_nk4.**odt**
   - http://temp.1060research.com/2010/09/hello_netkernel_nk4.**pdf**

7. *Computational results are resources and are identified within the address space*
   Example : In 'The Complete Works of Tom Geudens' (soon available in a shop near you) you'll find the 'Hello NetKernel' book.

16 According to my dictionary this means *bringing into being* or *turning concrete*.
17 Again according to my dictionary this is either a difficult mathematical concept or just means ... *mapping*

Firstmodule revisited

The possible actions with *firstmodule* were rather limited (there was only one action). We are going to extend that.

Where to make the changes

If you did not set up version control (shame on you, you should) you can make the changes in the current *firstmodule* directory (if you followed the book so far that is *urn.org.netkernelbook.tutorial.firstmodule-1.0.0* in *[install]/project-modules*).

Otherwise, make a new release in your repository (1.0.1 for example) and check it out to a new directory underneath project-modules. Like this for example :

```
youruser@yourmachine:~$ svn copy \
file:///svn/project-
modules/urn.org.netkernelbook.tutorial.firstmodule/trunk/src \
file:///svn/project-
modules/urn.org.netkernelbook.tutorial.firstmodule/release/1.0.1 \
-m "Create release 1.0.1"

Committed revision 8.

youruser@yourmachine:~$ svn checkout \
file:///svn/project-
modules/urn.org.netkernelbook.tutorial.firstmodule/release/1.0.1 \
/usr/NK4/project-
modules/urn.org.netkernelbook.tutorial.firstmodule-1.0.1
```

Remember that you also have to change the entry for *firstmodule* in *[install]/etc/modules.xml* if you want the new version to be the active one :

```
<module runlevel="7">project-
modules/urn.org.netkernelbook.tutorial.firstmodule-1.0.1/</module>
```

Modifications

There are a couple of ways to provide new functionality in *firstmodule*. We could define a new *rootspace*. We could define a new *entrypoint* in the existing mapper-overlay. We could do both. What we are going to do however is make the *grammar* in the existing *entrypoint* somewhat more dynamic.

My goal is for *firstmodule* to handle all resource requests like
res:/netkernelbook/firstmodule/hello
res:/netkernelbook/firstmodule/goodbye
res:/netkernelbook/firstmodule/jimcarrey
and so on and the *hello.dpml* program should return the corresponding html-resource (hello.html, goodbye.html, jimcarrey.html, ...).

State your goals **upfront**. Always ! In all the time I've worked with NetKernel, I've **never** been able to state an impossible goal. Some of them required asking others and redefining some of what I thought I knew of NetKernel, but impossible ? No. So don't let your goals be limited by what you (think you) know !

This is what the modified *grammar* could look like :

```
  <grammar>
      res:/netkernelbook/firstmodule/
      <group name="action">
        <regex>(hello|goodbye|jimcarrey)</regex>
      </group>
  </grammar>
```

The new thing here is the group-tag, which turns everything it matches/groups (in this case the last part of the resource request) into a ... resource that we can use in the request :

```
  <request>
    <identifier>active:dpml</identifier>
    <argument name="operator">
      res:/resources/dpml/hello.dpml
    </argument>
    <argument name="html">
      res:/resources/html/[[arg:action]].html
    </argument>
  </request>
```

As you can see we added one more argument in the request. It contains the uri of the html-resource that we want to see. Since we have it as an argument, *hello.dpml* no longer needs to hardcode it :

```
<sequence >
  <request assignment="response">
    <identifier>arg:html</identifier>
  </request>
</sequence>
```

As you can see, arguments passed to **active:dpml** can be used in the dpml code with the **arg:** scheme as well.

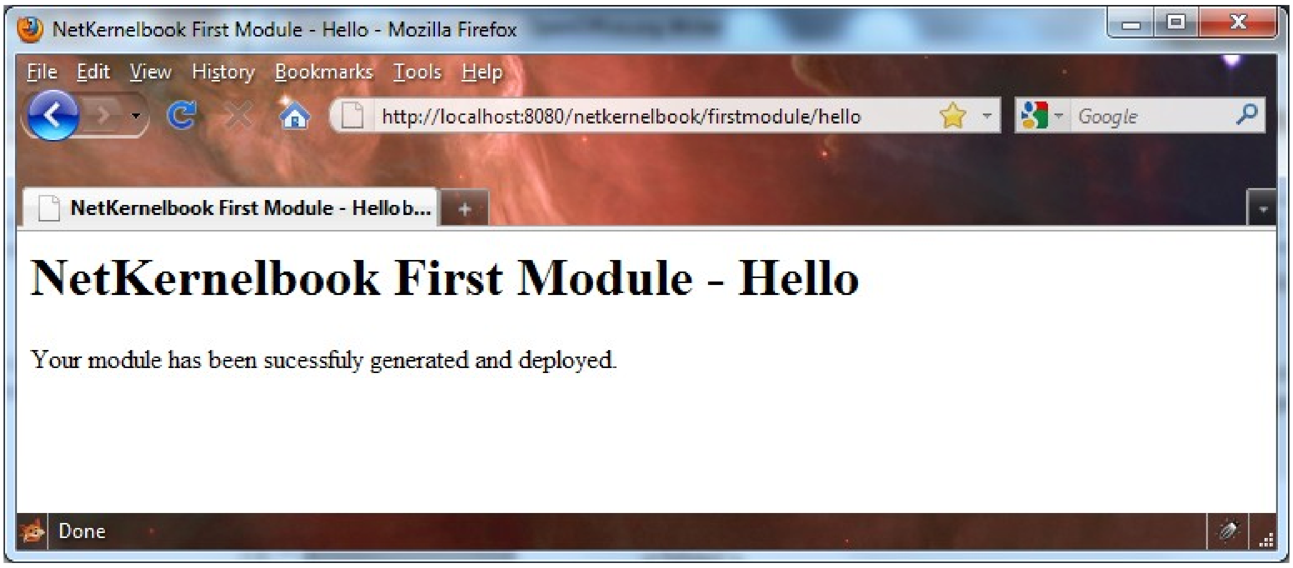
All that is left to do is to provide two new html files in *resources/html*. Here's *goodbye.html* :

```
<html>
  <head>
   <title>NetKernelbook First Module - Goodbye</title>
  </head>
  <body>
    <h1>NetKernelbook First Module - Goodbye</h1>
    <p>Your module has been sucessfuly generated and deployed.</p>
  </body>
</html>
```

And here's *jimcarrey.html* :

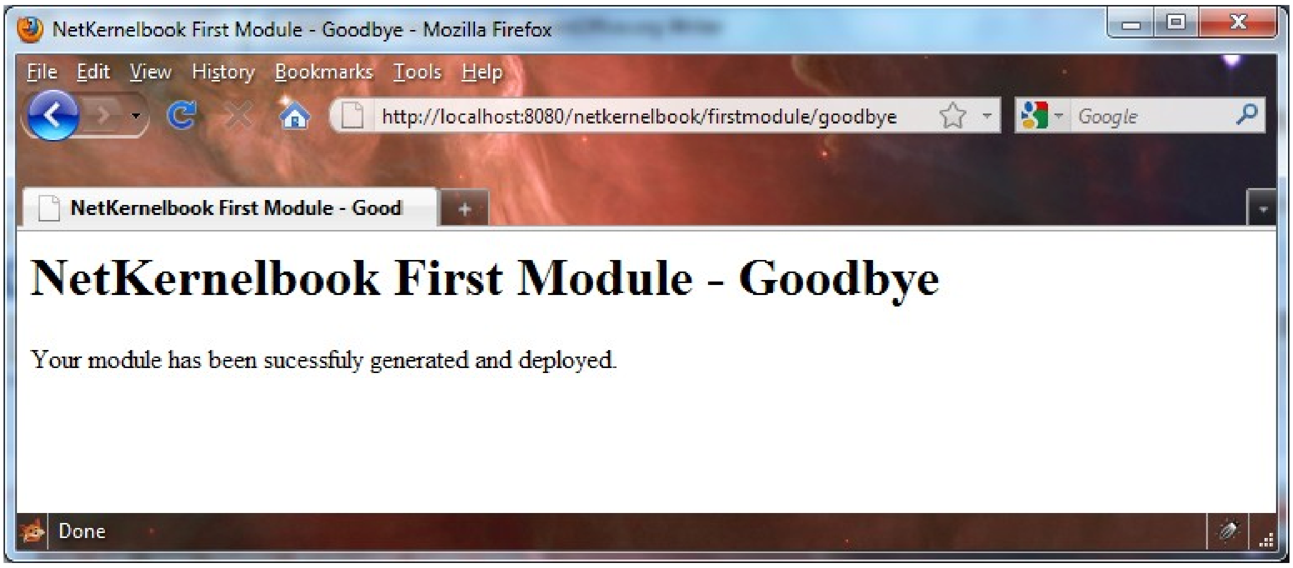
```
<html>
  <head>
   <title>NetKernelbook First Module - Jim Carrey</title>
  </head>
  <body>
    <h1>NetKernelbook First Module - Jim Carrey</h1>
    <p>And in case I do not see you any more ...
       good afternoon, good evening and goodbye !</p>
  </body>
</html>
```

And finally, here are the three possible outputs for **valid** requests to *firstmodule*


NetKernelbook First Module - Hello - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/netkernelbook/firstmodule/hello
Google
NetKernelbook First Module - Hellob...
Done

NetKernelbook First Module - Hello

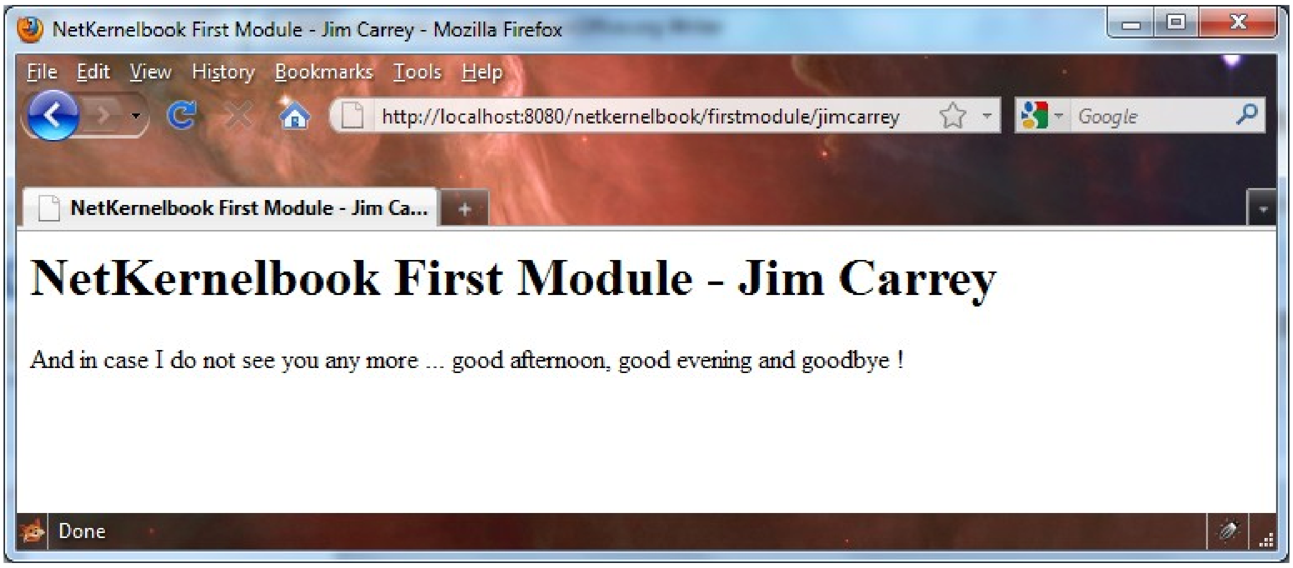
Your module has been sucessfuly generated and deployed.


NetKernelbook First Module - Goodbye - Mozilla Firefox

http://localhost:8080/netkernelbook/firstmodule/goodbye

NetKernelbook First Module - Goodbye

Your module has been sucessfuly generated and deployed.


NetKernelbook First Module - Jim Carrey - Mozilla Firefox

http://localhost:8080/netkernelbook/firstmodule/jimcarrey

NetKernelbook First Module - Jim Carrey

And in case I do not see you any more ... good afternoon, good evening and goodbye !

Version Control (only for those that have it)
This is a good moment to commit your changes to the repository. Like this for example :

```
youruser@yourmachine:~$ cd /usr/NK4/project-
modules/urn.org.netkernelbook.tutorial.firstmodule-1.0.1

youruser@yourmachine:~$ svn status
M       module.xml
?       resources/html/jimcarrey.html
?       resources/html/goodbye.html
M       resources/dpml/hello.dpml
```

```
youruser@yourmachine:~$ svn add \
 resources/html/jimcarrey.html resources/html/goodbye.html
A         resources/html/jimcarrey.html
A         resources/html/goodbye.html

youruser@yourmachine:~$ svn commit -m 'Release 1.0.1 complete'
Sending        module.xml
Sending        resources/dpml/hello.dpml
Adding         resources/html/goodbye.html
Adding         resources/html/jimcarrey.html
Transmitting file data ....
Committed revision 9.
```

Exercise
If I want to add a new html-resource, I have to modify the *module.xml* file. That is not completely what I stated in my goal ! Modify the grammar so that any html-resource that you put in resources/html can be shown by the *hello.dpml* program.

A first pattern – Webserver

NetKernel can do anything you imagine, but most of the time you're not the first person to think of a particular use[18]. A very common *pattern* is that of the webserver.

Goal
I use ExtJS[19] for most of my GUIs. You might use Dojo or JQuery or whatever, my point is that we have a library of files that we want to turn into *resources* and *serve*.

There is an extra requirement. Since I do not like to break what is working, I have at any given time multiple versions of ExtJS in action.

Requests will look like this
res:/ExtJS-3.0.3/ext-all.js
res:/ExtJS-3.1.0/adapter/ext/ext-base.js
res:/ExtJS-3.3.0/plugins/uxmedia.js

If you want to be able to follow this example all the way, you can find the relevant downloads at
http://www.sencha.com/products/js/thank-you.php?dl=extjs303
http://www.sencha.com/products/js/thank-you.php?dl=extjs310
http://www.sencha.com/products/js/thank-you.php?dl=extjs330

18 My apologies for hurting your ego.
19 http://www.sencha.com/products/js/

New module
You know by now how to set up a new module. I'm going to use this uri

```
urn:org:netkernelbook:extjs:server
```

and the corresponding directory will be

```
[install]/project-modules/urn.org.netkernelbook.extjs.server-1.0.0
```

And you do have a question of course, so let me answer that first ...

Do I want version control for this ?
Yes, you do want external libraries under your own version control. The two main reasons are

1. What's here and available on the internet today, might be gone tomorrow. Do you want to be the one to explain that a complete refactoring of your companies main application is necessary because you can no longer find the relevant libraries ? I think not !
2. You might want to do some customisation. No library is perfect, maybe there are community plugins available, ...

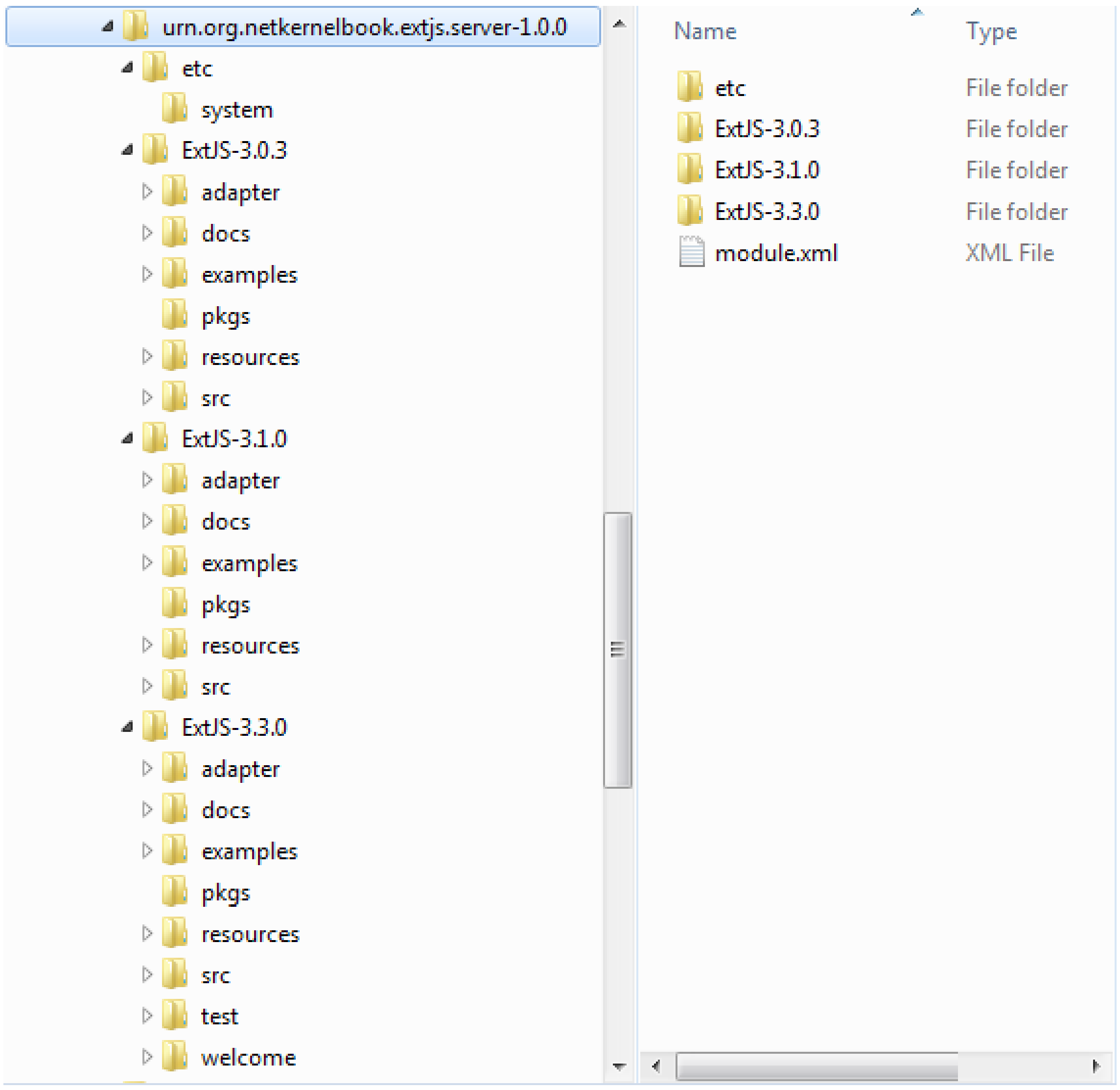
The best way to do this is to create the whole module as described below and *import* the whole shebang into your repository (you'll see in a minute that the module itself is very simple, there's no point in the initial trunk being empty). All plugins and customisations can then be added through *commits*.

You might by now feel that I'm a bit *fixated* on version control, backup and reproducibility of *resources*. Well, after having accidently dropped (as in *database drop*) an archive of about 300.000 mails and somewhat later sitting in on the meeting where the tape supplier declares that the tape that should contain the only backup actually contains *debris* ... your mind does wonderfuly get focused yes ...

The rootspace
This is the important part of the *module.xml* file for our extjs-server module

```
<rootspace>

  <fileset>
    <regex>res:/etc/system/SimpleDynamicImportHook.xml</regex>
  </fileset>

  <fileset>
    <regex>res:/ExtJS-[0-9]\.[0-9]\.[0-9]/.*</regex>
  </fileset>

</rootspace>
```

Yes, that is all. Like I said, very simple. We make the ExtJS files in our module available and also open them up to the HTTPFulcrum. This is the structure of our module :



```
urn.org.netkernelbook.extjs.server-1.0.0
    etc
        system
    ExtJS-3.0.3
        adapter
        docs
        examples
        pkgs
        resources
        src
    ExtJS-3.1.0
        adapter
        docs
        examples
        pkgs
        resources
        src
    ExtJS-3.3.0
        adapter
        docs
        examples
        pkgs
        resources
        src
        test
        welcome
```

Name	Type
etc	File folder
ExtJS-3.0.3	File folder
ExtJS-3.1.0	File folder
ExtJS-3.3.0	File folder
module.xml	XML File

The content of the ExtJS-x.x.x directories comes straight from the corresponding zip-files and the *SimpleDynamicImportHook.xml* file is exactly the same as in *firstmodule*.

Testing
http://localhost:8080/ExtJS-3.0.3/license.txt
http://localhost:8080/ExtJS-3.0.3/docs/welcome.html
http://localhost:8080/ExtJS-3.1.0/license.txt
http://localhost:8080/ExtJS-3.1.0/docs/welcome.html
http://localhost:8080/ExtJS-3.3.0/license.txt
http://localhost:8080/ExtJS-3.3.0/docs/welcome.html

There, that should be sufficient proof. It isn't you say ? You are right, lets modify *firstmodule* a bit so we have something to show for !

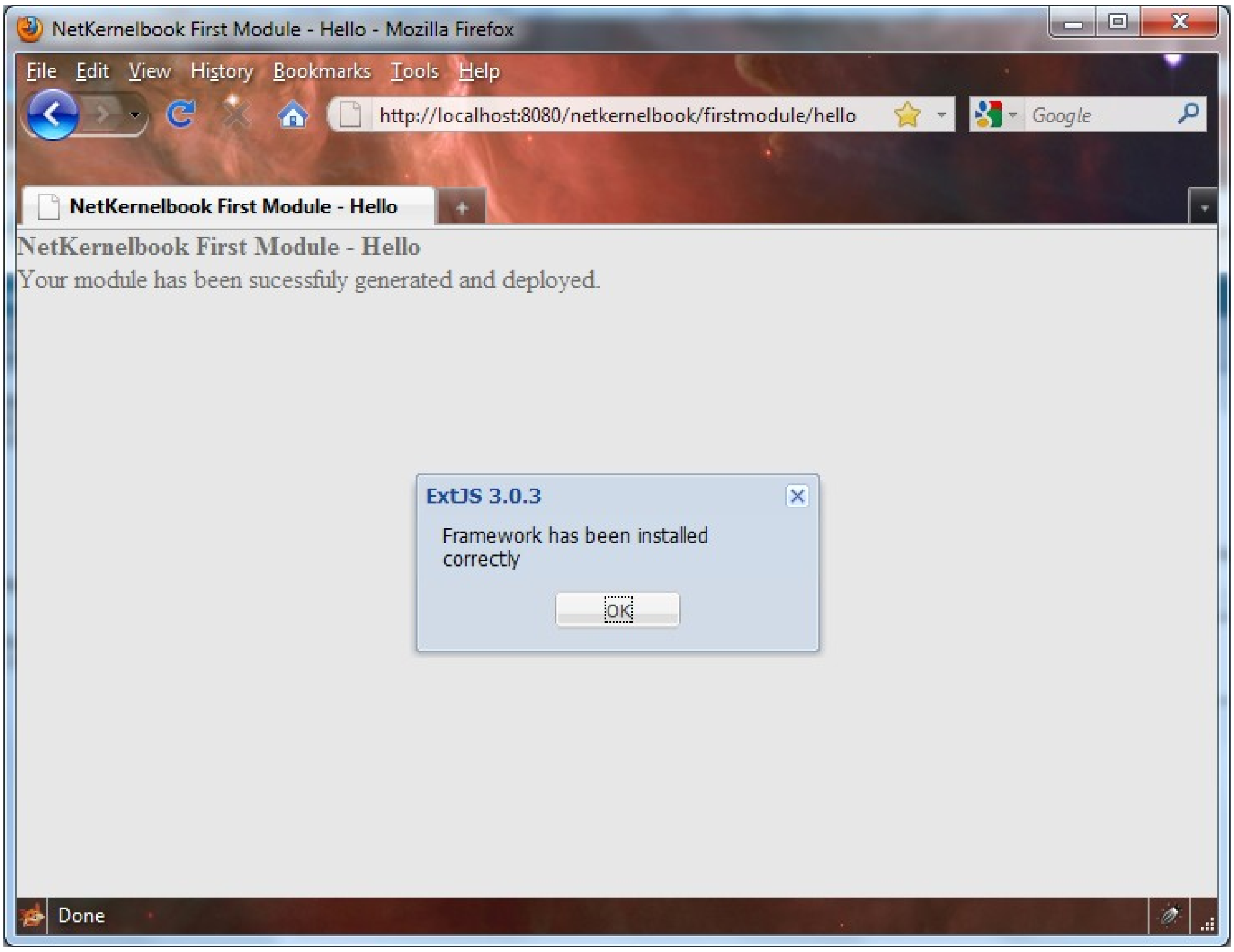
Changing firstmodule again
You can choose for yourself if you are going to do this in release 1.0.0 or 1.0.1 or if you are going to make a new 1.0.2.

What we are going to do is change *hello.html* (present in both releases) so it looks like this :

```
<html>
  <head>
   <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
   <link
     href="http://localhost:8080/ExtJS-3.0.3/resources/css/ext-all-notheme.css"
     rel="stylesheet" type="text/css" />
   <link
     href="http://localhost:8080/ExtJS-3.0.3/resources/css/xtheme-blue.css"
     rel="stylesheet" type="text/css" />
   <title>NetKernelbook First Module - Hello</title>
  </head>

  <body>
    <h1>NetKernelbook First Module - Hello</h1>
    <p>Your module has been sucessfuly generated and deployed.</p>
    <script type="text/javascript"
      src="http://localhost:8080/ExtJS-3.0.3/adapter/ext/ext-base.js">
      /* <![CDATA[ */
      /* ]]> */
    </script>
    <script type="text/javascript"
      src="http://localhost:8080/ExtJS-3.0.3/ext-all.js">
      /* <![CDATA[ */
      /* ]]> */
    </script>
    <script type="text/javascript">
      /* <![CDATA[ */
      Ext.namespace('test');

      test.app = function() {
        // private space

        // public space
        return {
          init: function() {
            Ext.Msg.alert('ExtJS 3.0.3',
              'Framework has been installed correctly');
          }
        };
      }();
      /* ]]> */
    </script>
    <script type="text/javascript">
      /* <![CDATA[ */
      Ext.onReady(test.app.init, test.app);
      /* ]]> */
    </script>
  </body>
</html>
```

My apologies for reducing the fontsize on that, but it looked pretty bad otherwise. Don't be daunted by this, we are including the main ExtJS stylesheets, the main ExtJS javascriptfiles and I define one javascript function which I then execute when the page is fully loaded.

When you now try http://localhost:8080/netkernelbook/firstmodule/hello, you should see something similar to this


NetKernelbook First Module - Hello - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/netkernelbook/firstmodule/hello
Google
NetKernelbook First Module - Hello
Done

NetKernelbook First Module - Hello

Your module has been sucessfuly generated and deployed.

ExtJS 3.0.3

Framework has been installed correctly

OK

Wonderful, but ...
You'll agree with me that ExtJS is indeed being served by NetKernel. You'll also agree with me that having to hardcode *http://localhost:8080* before every resource-request in the html is not very flexible.

There is a solution for that. Since *hello.html* is also a resource, we can have it pre-processed by XRL. We'll check out the **XML Recursion Language** in detail in another chapter, but we are going to make use of it now.

First we import it in the space-tag of *firstmodule*'s *module.xml* file

```
  <import>
    <private />
    <uri>urn:org:netkernel:lang:xrl</uri>
  </import>
```

And then we can modify *hello.html* again

```
<html xmlns:xrl="http://netkernel.org/xrl">
  <head>
   <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
   <link href="/ExtJS-3.0.3/resources/css/ext-all-notheme.css"
     xrl:resolve="href"
     rel="stylesheet" type="text/css" />
   <link href="/ExtJS-3.0.3/resources/css/xtheme-blue.css"
     xrl:resolve="href"
     rel="stylesheet" type="text/css" />
   <title>NetKernelbook First Module - Hello</title>
  </head>
  <body>
    <h1>NetKernelbook First Module - Hello</h1>
    <p>Your module has been sucessfuly generated and deployed.</p>
    <script type="text/javascript"
      src="/ExtJS-3.0.3/adapter/ext/ext-base.js"
      xrl:resolve="src">
      /* <![CDATA[ */
      /* ]]> */
    </script>
    <script type="text/javascript" src="/ExtJS-3.0.3/ext-all.js"
      xrl:resolve="src">
      /* <![CDATA[ */
      /* ]]> */
    </script>
    <script type="text/javascript">
      /* <![CDATA[ */
      Ext.namespace('test');

      test.app = function() {
        // private space

        // public space
        return {
          init: function() {
            Ext.Msg.alert('ExtJS 3.0.3',
             'Framework has been installed correctly');
          }
        };
      }();
      /* ]]> */
    </script>
    <script type="text/javascript">
      /* <![CDATA[ */
      Ext.onReady(test.app.init, test.app);
      /* ]]> */
    </script>
  </body>
</html>
```

Note the *xrl:resolve* tags and the difference in the *href* and *src* tags. Now, that looks a whole lot better (more portable), does it not ?

Conclusion

In this chapter we cut out the heart and soul of *firstmodule* and it did have some surprises in store for us. We talked about (root)spaces, endpoints, requests and grammars (not necessarily in that order). Next we made *firstmodule* a bit more flexible and we finished by using NetKernel as a file-/webserver.

If you came out of this chapter with your head still on, you're now ready to do some experimenting of your own. By all means do and have some fun while you're at it ! The next chapter will be there, waiting for you when you come back.

CHAPTER 4

Humongous Fun

<chapter 4 text/>

Getting and Installing NetKernel

Prerequisites

To run NetKernel you must have a computer and operating system capable of running Java 1.5 or 1.6. NetKernel is platform neutral and has been deployed successfully on Windows 2000, Windows 7, Windows XP, Windows Vista, Windows Server 2003, Apple Mac OS X, Linux (Redhat, Suse, Debian, Ubuntu) and Solaris.

The above comes straight from the install notes. I just want to add that although a JRE (java runtime environment) is sufficient to run NetKernel with all its features, I strongly advise you to install a JDK on machines where you do NetKernel development.

Download

This book deals with the (open source) NetKernel Standard Edition and the versions for that can be found under http://download.netkernel.org/nkse/ :
- Select the download for the *4.1.x*. version.
- Pick a mirror.
- Save the *1060-NetKernel-4.1.x.jar* file to your system.
- While the download is running, read the *install notes*.

Installation

Installation is very easy and pretty much identical on any platform. Below you'll find the transcripts of an installation on **Windows 7** and **Ubuntu 10.04 LTS - the Lucid Lynx**.

Running the downloaded jar – Windows

First position yourself in the directory above the one where you want to install NetKernel (I'm going to install in D:\NK4, so I position in D:, I also put the downloaded jar there for ease of use).

```
C:\Users\your_user>d:
D:\>java -jar 1060-NetKernel-SE-4.1.1.jar
Expanding urn.com.ten60.core.boot-1.13.22
Expanding urn.com.ten60.core.cache.se-1.2.11
Expanding urn.com.ten60.core.layer0-1.31.57
Expanding urn.com.ten60.core.module.standard-1.20.29
Expanding urn.com.ten60.core.netkernel.api-4.1.5
Expanding urn.com.ten60.core.netkernel.impl-4.13.24
I 17:11:40 Kernel
Starting 1060-NetKernel-SE
...
```

```
I 17:11:45 Kernel        NetKernel Ready, accepting requests...
I 17:11:45 ModuleManager System now at RunLevel [2]
***************************************************************
* JAR BOOT NOTES
* ---------------
* NetKernel is now running an HTTP server on port 1060
*
* To start using NetKernel open a web browser
* and go to:   http://localhost:1060/
***************************************************************
```

Running the downloaded jar - Ubuntu
Starting the downloaded jar on Linux is exactly the same as on Windows, but we are going to do a bit of preparation in advance, this will make things easier later on.

```
your_user@ubuntumachine:~$ sudo groupadd --gid 1060 dexter
your_user@ubuntumachine:~$ sudo useradd --uid 1060 --gid 1060 -m \
 -d /home/dexter -s /bin/bash -c 'NetKernel software' dexter
your_user@ubuntumachine:~$ sudo passwd dexter
your_user@ubuntumachine:~$ sudo mkdir /usr/NK4
your_user@ubuntumachine:~$ sudo chown dexter:dexter /usr/NK4
```

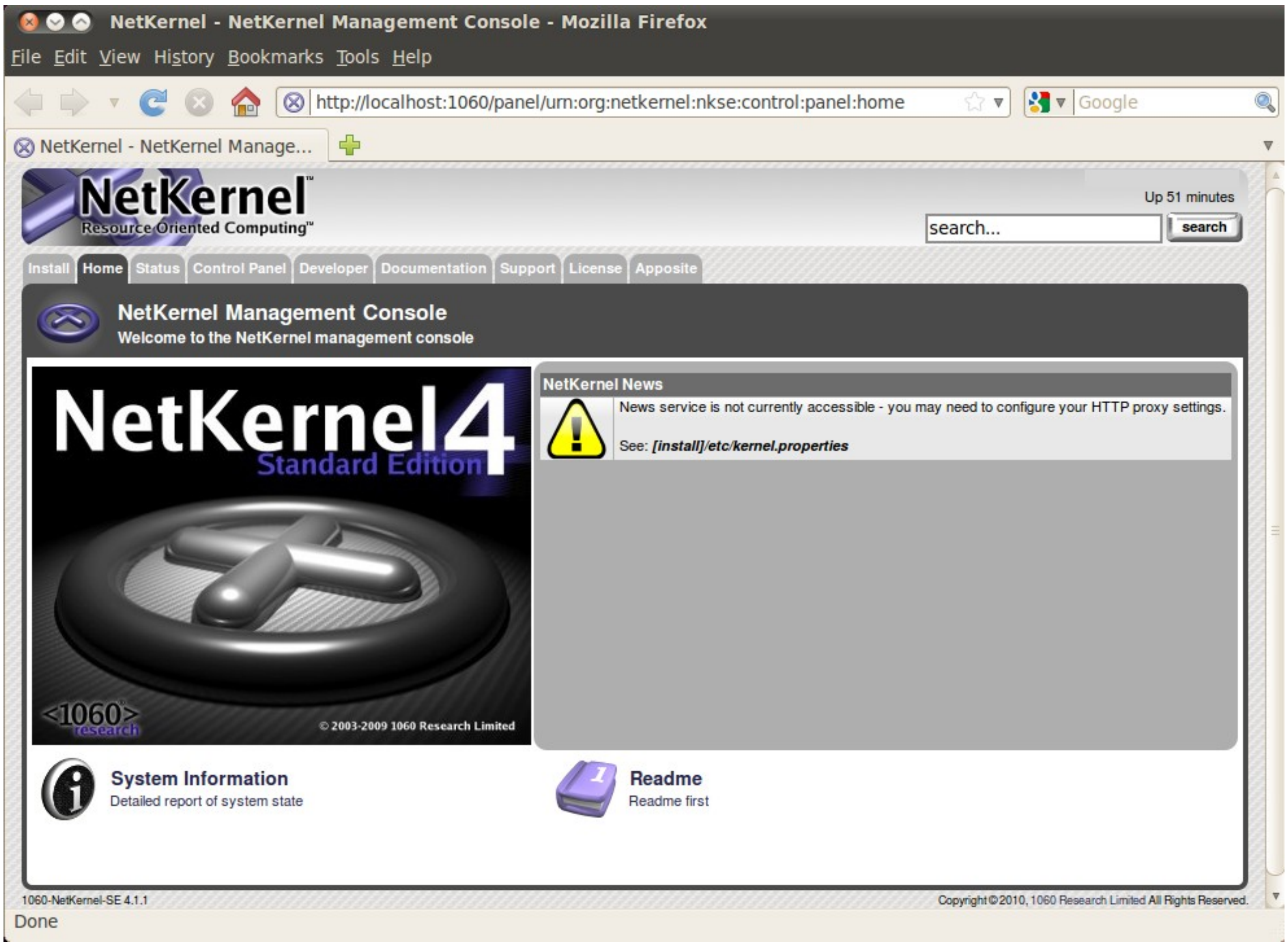
You can of course pick your own groupname, username and directorynames, the rest of this installation procedure will go with the values used above.

Change to the newly created user, position yourself in the directory above the one where you want to install NetKernel and start the downloaded jar.

```
dexter@ubuntumachine:~$ cd /usr
dexter@ubuntumachine:/usr$ java -jar 1060-NetKernel-SE-4.1.1.jar
Expanding urn.com.ten60.core.boot-1.13.22
Expanding urn.com.ten60.core.cache.se-1.2.11
Expanding urn.com.ten60.core.layer0-1.31.57
Expanding urn.com.ten60.core.module.standard-1.20.29
Expanding urn.com.ten60.core.netkernel.api-4.1.5
Expanding urn.com.ten60.core.netkernel.impl-4.13.24
I 17:22:40 Kernel
Starting 1060-NetKernel-SE
...I 17:22:42 Kernel        NetKernel Ready, accepting requests...
I 17:22:42 ModuleManager System now at RunLevel [2]
***************************************************************
* JAR BOOT NOTES
* ---------------
* NetKernel is now running an HTTP server on port 1060
*
* To start using NetKernel open a web browser
* and go to:   http://localhost:1060/
***************************************************************
```

Verification – all environments
If all is well, you can now :
- fire up your favorite webbrowser
- enter http://localhost:1060

And you should get this screen :


NetKernel - NetKernel Management Console - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:1060/panel/urn:org:netkernel:nkse:control:panel:home
Google
NetKernel - NetKernel Manage...
NetKernel Resource Oriented Computing
Up 51 minutes
search...
search
Install Home Status Control Panel Developer Documentation Support License Apposite
NetKernel Management Console
Welcome to the NetKernel management console
NetKernel4 Standard Edition
<1060> research
© 2003-2009 1060 Research Limited
NetKernel News
System Information
Detailed report of system state
Readme
Readme first
1060-NetKernel-SE 4.1.1
Copyright © 2010, 1060 Research Limited All Rights Reserved.
Done

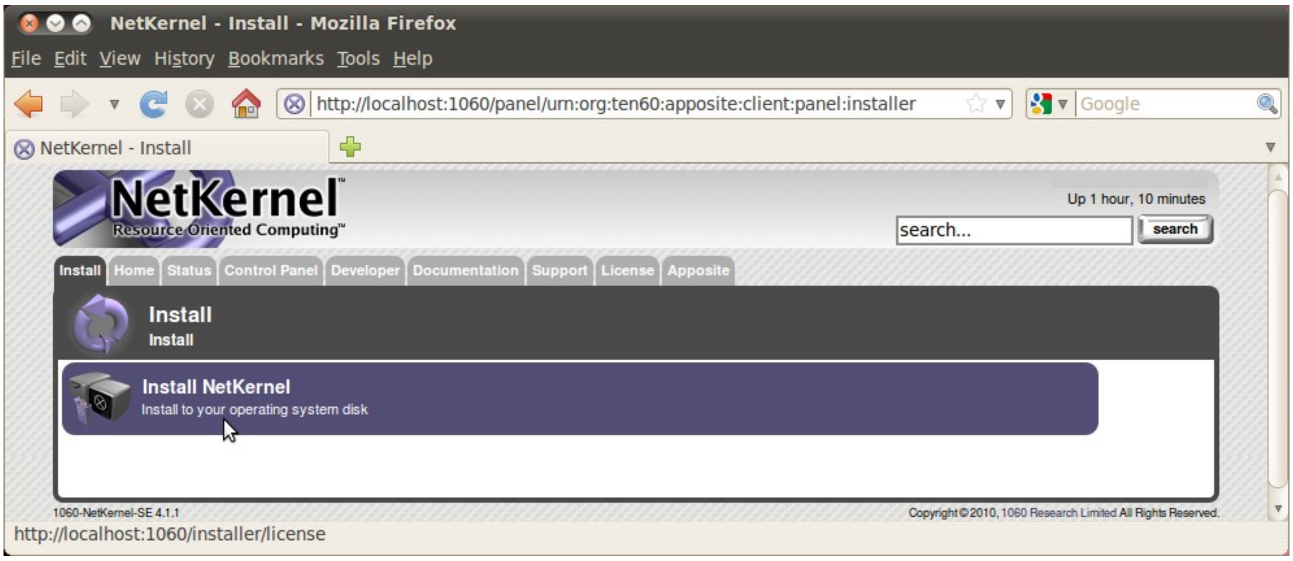
News service is not currently accessible - you may need to configure your HTTP proxy settings.

See: *[install]/etc/kernel.properties*

If your machine has internet access you will get news items underneath *NetKernel News*.

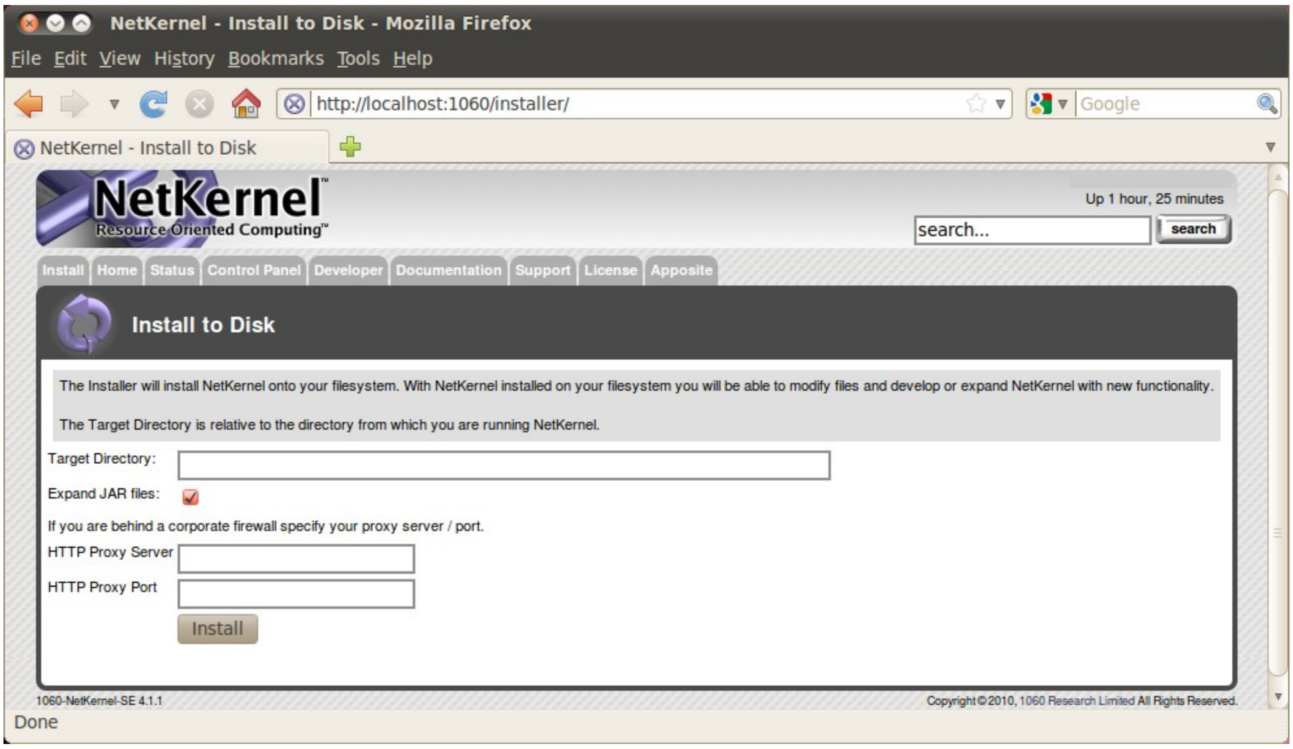
The NetKernel interface will perform well on any **modern up-to-date** webbrowser.

Installation – all environments
By all means, browse through the tabs and check stuff (you probably did not read that *Readme first*, did you ?). When you're ready for the installation to disk, select the *Install* tab.


NetKernel - Install - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:1060/panel/urn:org:ten60:apposite:client:panel:installer
NetKernel - Install
NetKernel Resource Oriented Computing
Up 1 hour, 10 minutes
search...
search
Install Home Status Control Panel Developer Documentation Support License Apposite
Install
Install
Install NetKernel
Install to your operating system disk
1060-NetKernel-SE 4.1.1
Copyright © 2010, 1060 Research Limited All Rights Reserved.
http://localhost:1060/installer/license

I do wonder what that option does. Lets find out and press it !

Do read the license you'll see next. This book is concerned only with the (open source) NetKernel Standard Edition. **If you cannot comply with the public license terms you must obtain a commercial license from www.1060research.com**.


NetKernel - Install to Disk - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:1060/installer/
NetKernel - Install to Disk
NetKernel Resource Oriented Computing
Up 1 hour, 25 minutes
search...
search
Install Home Status Control Panel Developer Documentation Support License Apposite
Install to Disk

The Installer will install NetKernel onto your filesystem. With NetKernel installed on your filesystem you will be able to modify files and develop or expand NetKernel with new functionality.

The Target Directory is relative to the directory from which you are running NetKernel.

Target Directory:
Expand JAR files: ✓
If you are behind a corporate firewall specify your proxy server / port.
HTTP Proxy Server
HTTP Proxy Port

Install
1060-NetKernel-SE 4.1.1
Copyright © 2010, 1060 Research Limited All Rights Reserved.
Done

All the earlier positioning pays of here, for by entering NK4 in the *Target Directory* field the installation will go where I want it (as well on Windows as on Linux).

Press *Install* ...

Verification – Windows
The following message will show :
NetKernel was successfully installed onto your filesystem at ***D:\\NK4***

Check this visually, there should be five new subdirectories underneath D:\NK4.

```
D:\NK4>dir
 Volume in drive D is xxxx
 Volume Serial Number is xxxx-xxxx

 Directory of D:\NK4

04/09/2010  21:18    <DIR>          .
04/09/2010  21:18    <DIR>          ..
04/09/2010  21:18    <DIR>          bin
04/09/2010  21:18    <DIR>          etc
04/09/2010  21:18    <DIR>          lib
04/09/2010  21:18    <DIR>          log
04/09/2010  21:18    <DIR>          modules
               0 File(s)              0 bytes
               7 Dir(s)  x bytes free
```

Verification – Ubuntu
The following message will show :
NetKernel was successfully installed onto your filesystem at ***/usr/NK4***

Check this visually, there should be five new subdirectories underneath /usr/NK4.

```
dexter@ubuntumachine:/usr/NK4$ ls -la
total 28
drwxr-xr-x  7 dexter dexter 4096 2010-09-04 21:16 .
drwxr-xr-x 14 root   root   4096 2010-09-04 17:20 ..
drwxr-xr-x  2 dexter dexter 4096 2010-09-04 21:16 bin
drwxr-xr-x  4 dexter dexter 4096 2010-09-04 21:16 etc
drwxr-xr-x  4 dexter dexter 4096 2010-09-04 21:16 lib
drwxr-xr-x  2 dexter dexter 4096 2010-09-04 21:16 log
drwxr-xr-x 36 dexter dexter 4096 2010-09-04 21:16 modules
```

Stopping downloaded jar – all environments

You are now almost ready for your first run. Press *CTRL-C* in the window where you are running the downloaded jar. This will stop the installation-run.

```
...
^CI 16:30:58 Kernel        NetKernel Pausing, flushing pending
requests, new requests queued...
I 16:30:58 HTTPTranspor~ Decommissioning HTTP Transport
I 16:30:58 HTTPTranspor~ Graceful shutdown {}
```

First run from disk – Windows

```
C:\Users\your_user>d:
D:\>cd NK4
D:\NK4>bin\netkernel.bat
```

```
I 16:44:58 Kernel
Starting 1060-NetKernel-SE
Resource Oriented Computing Platform
Version 4.1.1
...
I 16:45:29 Kernel        NetKernel Ready, accepting requests...
I 16:45:29 ModuleManager System now at RunLevel [7]
I 16:45:29 InitEndpoint  Init completed - system at RunLevel [7]
I 16:45:29 CronTransport Added Job [Apposite Synchronize @ Every
3rd Day] of type [crontab]
```

First run from disk – Ubuntu

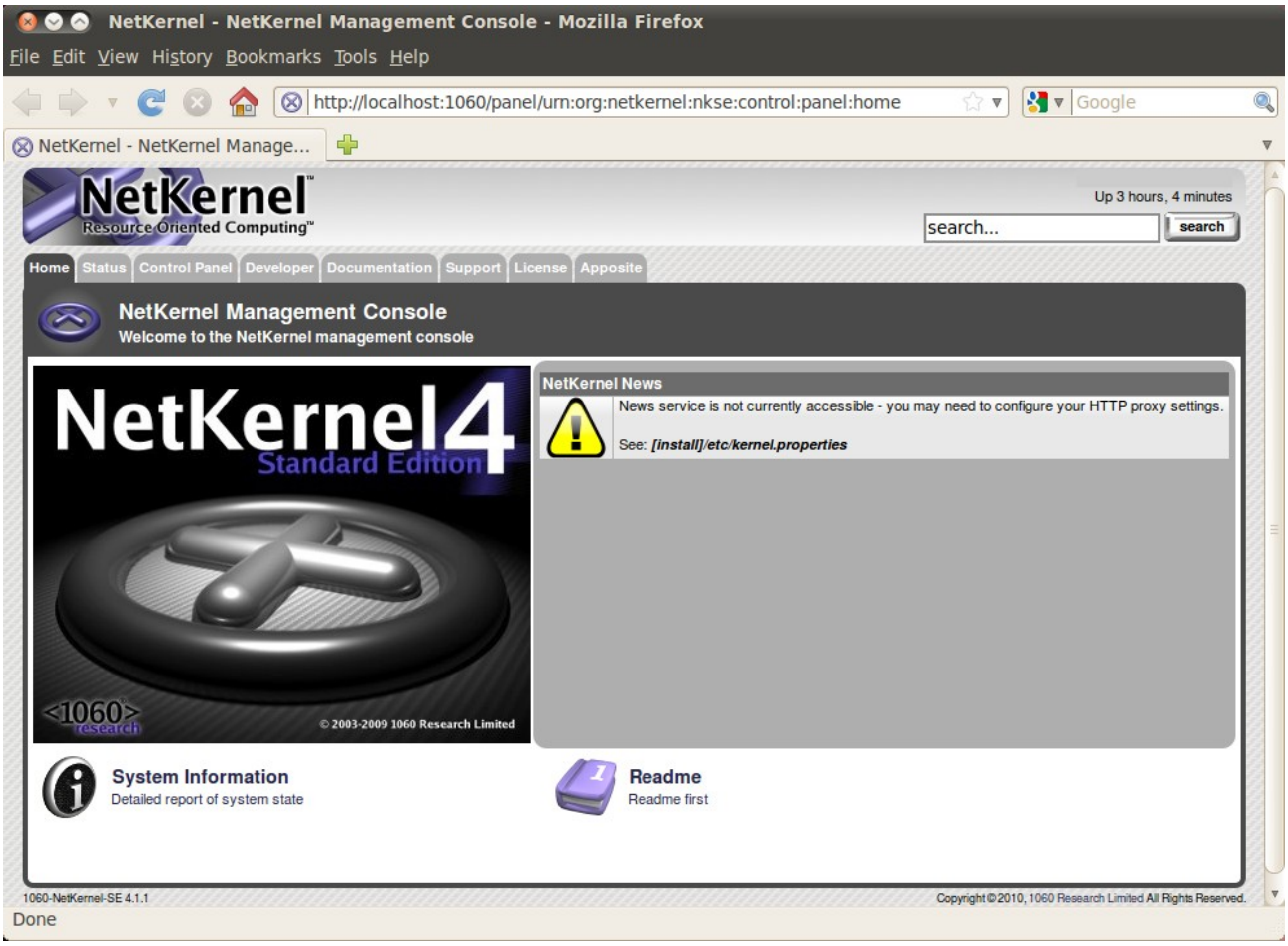
```
dexter@ubuntumachine:~$ cd /usr/NK4
dexter@ubuntumachine:~$ bin/netkernel.sh
```

```
I 17:40:15 Kernel
Starting 1060-NetKernel-SE
Resource Oriented Computing Platform
Version 4.1.1
...
I 17:41:08 Kernel        NetKernel Ready, accepting requests...
I 17:41:08 ModuleManager System now at RunLevel [7]
I 17:41:08 InitEndpoint  Init completed - system at RunLevel [7]
I 17:41:08 CronTransport Added Job [Apposite Synchronize @ Every
3rd Day] of type [crontab]
```

Verification – all environments

If all is well, you can now once again :
- fire up your favorite webbrowser
- enter http://localhost:1060

And you should get this screen :


NetKernel - NetKernel Management Console - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:1060/panel/urn:org:netkernel:nkse:control:panel:home
NetKernel - NetKernel Manage...
NetKernel Resource Oriented Computing
Up 3 hours, 4 minutes
search...
search
Home Status Control Panel Developer Documentation Support License Apposite
NetKernel Management Console
Welcome to the NetKernel management console
NetKernel4 Standard Edition
© 2003-2009 1060 Research Limited
NetKernel News
System Information
Detailed report of system state
Readme
Readme first
1060-NetKernel-SE 4.1.1
Copyright © 2010, 1060 Research Limited All Rights Reserved.
Done

News service is not currently accessible - you may need to configure your HTTP proxy settings.

See: *[install]/etc/kernel.properties*

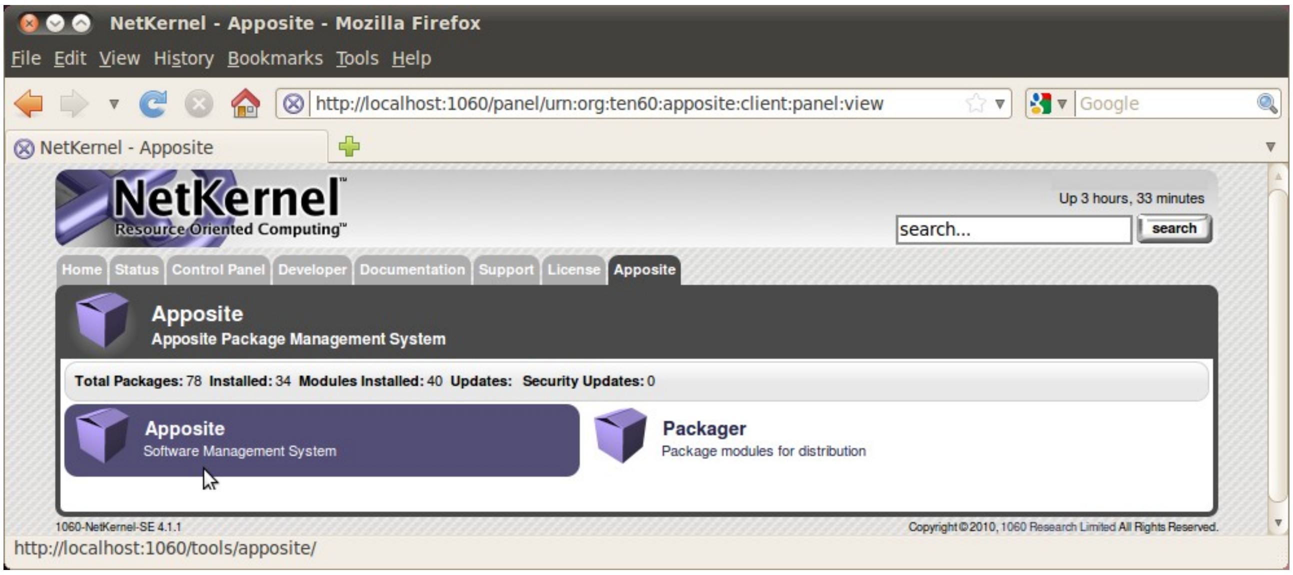
The only visible difference with the NetKernel Management Console we saw earlier is that the *Install* tab is no longer there.

Apposite – all environments

Before you do anything else, you should update the current NetKernel modules, to make sure you have all security and other patches. NetKernel has a *Software Management System* called *Apposite* to take care of this. In fact, *Apposite* itself is managed and updated this way, as is every part of NetKernel.

The default Base URI for the Apposite repository is http://apposite.netkernel.org/repo/. If your NetKernel instance does not have access to the internet, you'll not be able to reach this. In that case you should **first** set up your own. Appendix B explains how to do this. Only then continue with the remainder of Appendix A.

Select the *Apposite* tab.



NetKernel - Apposite - Mozilla Firefox

http://localhost:1060/panel/urn:org:ten60:apposite:client:panel:view

Up 3 hours, 33 minutes

Home | Status | Control Panel | Developer | Documentation | Support | License | Apposite

Apposite
Apposite Package Management System

Total Packages: 78 Installed: 34 Modules Installed: 40 Updates: Security Updates: 0

Apposite – Software Management System

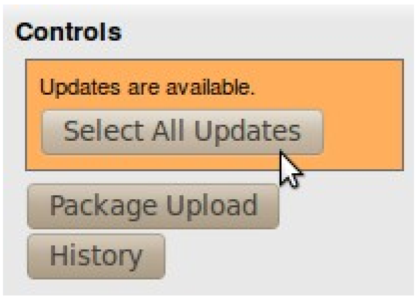
Packager – Package modules for distribution

1060-NetKernel-SE 4.1.1

Copyright © 2010, 1060 Research Limited All Rights Reserved.

http://localhost:1060/tools/apposite/

Packaging is discussed elsewhere in this book. Select and press *Apposite*.

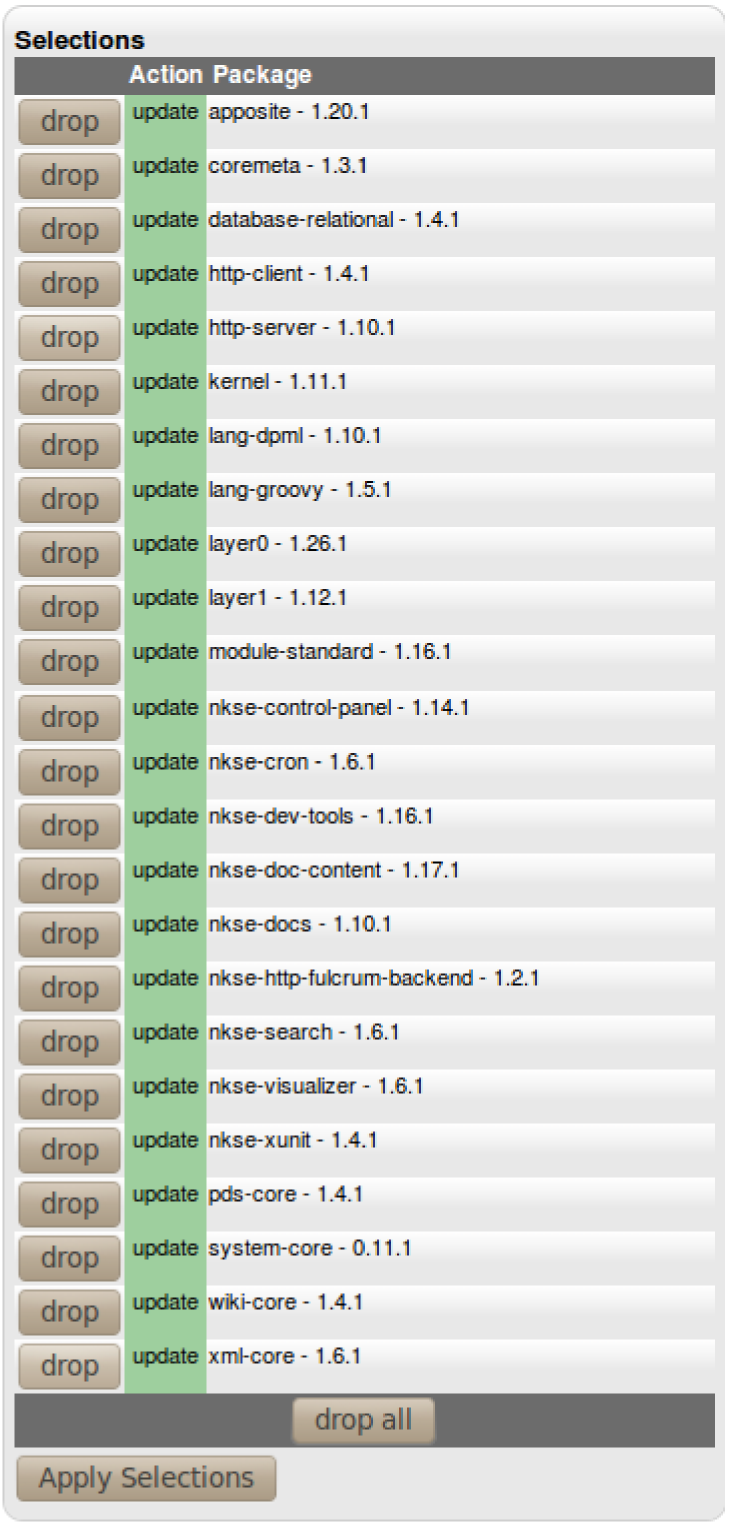


Controls

Updates are available.

You should see orange ! There should be updates available, very likely (you can see this if you scroll down the page) for *Apposite* itself. If there are no updates available at this point ... something went wrong in an earlier step.

The action to take suggests itself rather clearly ... press *Select All Updates*.

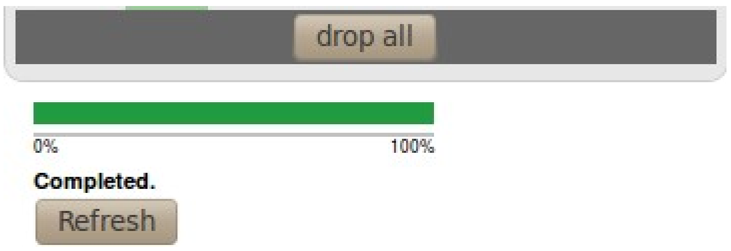


Selections

	Action	Package
drop	update	apposite - 1.20.1
drop	update	coremeta - 1.3.1
drop	update	database-relational - 1.4.1
drop	update	http-client - 1.4.1
drop	update	http-server - 1.10.1
drop	update	kernel - 1.11.1
drop	update	lang-dpml - 1.10.1
drop	update	lang-groovy - 1.5.1
drop	update	layer0 - 1.26.1
drop	update	layer1 - 1.12.1
drop	update	module-standard - 1.16.1
drop	update	nkse-control-panel - 1.14.1
drop	update	nkse-cron - 1.6.1
drop	update	nkse-dev-tools - 1.16.1
drop	update	nkse-doc-content - 1.17.1
drop	update	nkse-docs - 1.10.1
drop	update	nkse-http-fulcrum-backend - 1.2.1
drop	update	nkse-search - 1.6.1
drop	update	nkse-visualizer - 1.6.1
drop	update	nkse-xunit - 1.4.1
drop	update	pds-core - 1.4.1
drop	update	system-core - 0.11.1
drop	update	wiki-core - 1.4.1
drop	update	xml-core - 1.6.1

A *Selections* list will appear. You can see the list I get for version 4.1.1. above, yours may differ. I do know that you immediately want to add other stuff (Python for example) as well, but **don't** ! Take the logical next step, press *Apply Selections*.

Be patient, depending on which repository you use this make take a minute or so. Underneath the Selections list you'll get an update of what's going on. When finished you'll see a *Refresh* button appear there ... like this :



Guess what you have to do next. That's right ... press *Refresh*. If all goes well you should get the *Apposite* screen back, with all updated packages showing their new version number and all orange gone !

Conclusion

Installing NetKernel is – considering what you get in return – pretty simple and uniform across platforms. For a production system you might want to run NetKernel as a service or a daemon that gets started at boottime. Appendix C deals with setting that up.

A P P E N D I X B

Setting up your own Apposite Repository

Prerequisites

There is a bit of a 'chicken and the egg'-problem[20] here. The reason you would need your own Apposite Repository is that you do or will not allow your NetKernel instance access to the internet. However, in order to set up your own Apposite Repository, you will need access to the internet. No way around it I'm afraid. It does however not have to be from the machine you run your NetKernel instance on !

I'll discuss a setup via the **rsync-**utility on **Windows 7** and **Ubuntu 10.04 LTS - the Lucid Lynx**. For Ubuntu this utility is present by default, for Windows 7 we'll use the one available in the **Cygwin** package. Don't worry if Cygwin means nothing to you, I'll discuss the setup for that as well.

In fact, that's what I'm going to do first ...

Yes, I do know that there are other rsync ports available for Windows. Feel *free* to use them ... most of them are not (free to use that is). Some of the others are limited to specific usages. Trust me, it will do you no harm to have a Linux-like shell with lots of Linux-utilities available on your Windows machine. You can thank me later !

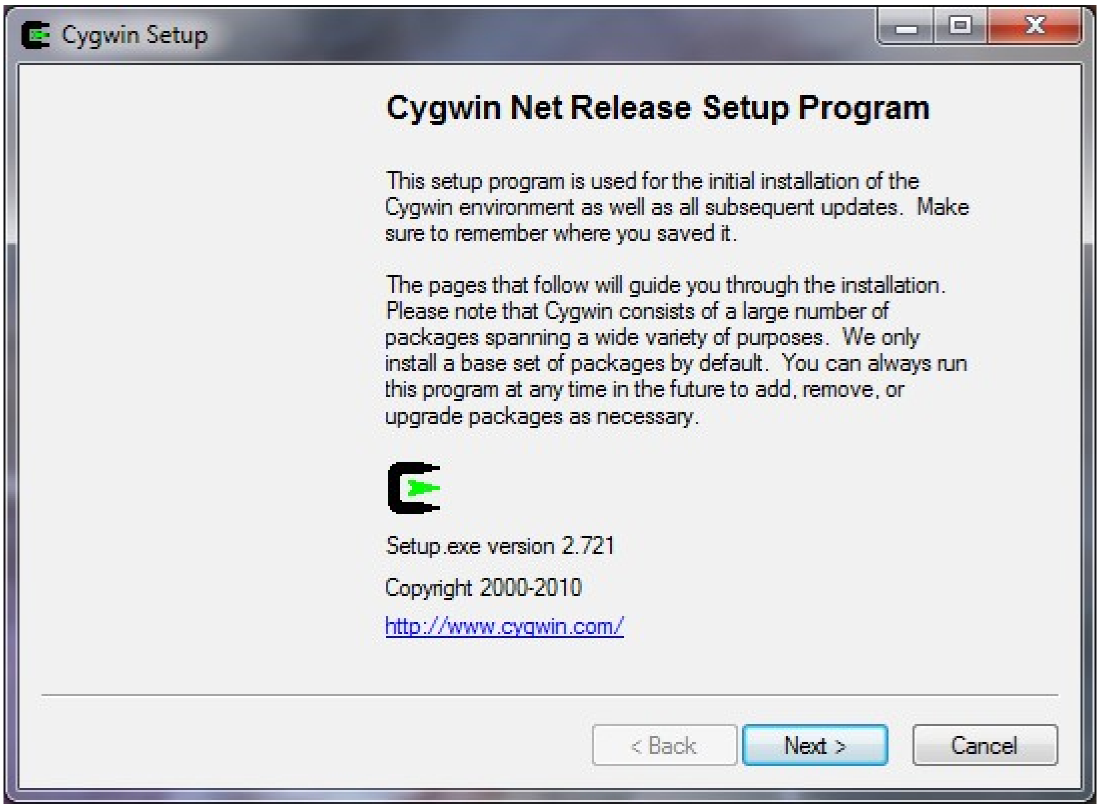
Preparation

Getting Cygwin – Windows 7 only

You can get the Cygwin setup file at http://www.cygwin.com/setup.exe. Download *it*. Note that not only the first install is done with this file, but also all subsequent updates (or installation of new utilities you may require). So download it to a place where you can find it again (I keep it on my desktop in fact).

20 That one has been solved by science, the chicken came first. Something to do with a certain protein.

Installing Cygwin – Windows 7 only
Start the dowloaded *setup.exe*.


Cygwin Net Release Setup Program

This setup program is used for the initial installation of the Cygwin environment as well as all subsequent updates. Make sure to remember where you saved it.

The pages that follow will guide you through the installation. Please note that Cygwin consists of a large number of packages spanning a wide variety of purposes. We only install a base set of packages by default. You can always run this program at any time in the future to add, remove, or upgrade packages as necessary.

Setup.exe version 2.721

Copyright 2000-2010

http://www.cygwin.com/

Read *the text* (told you about keeping the setup.exe, didn't I ?)
Press *Next*

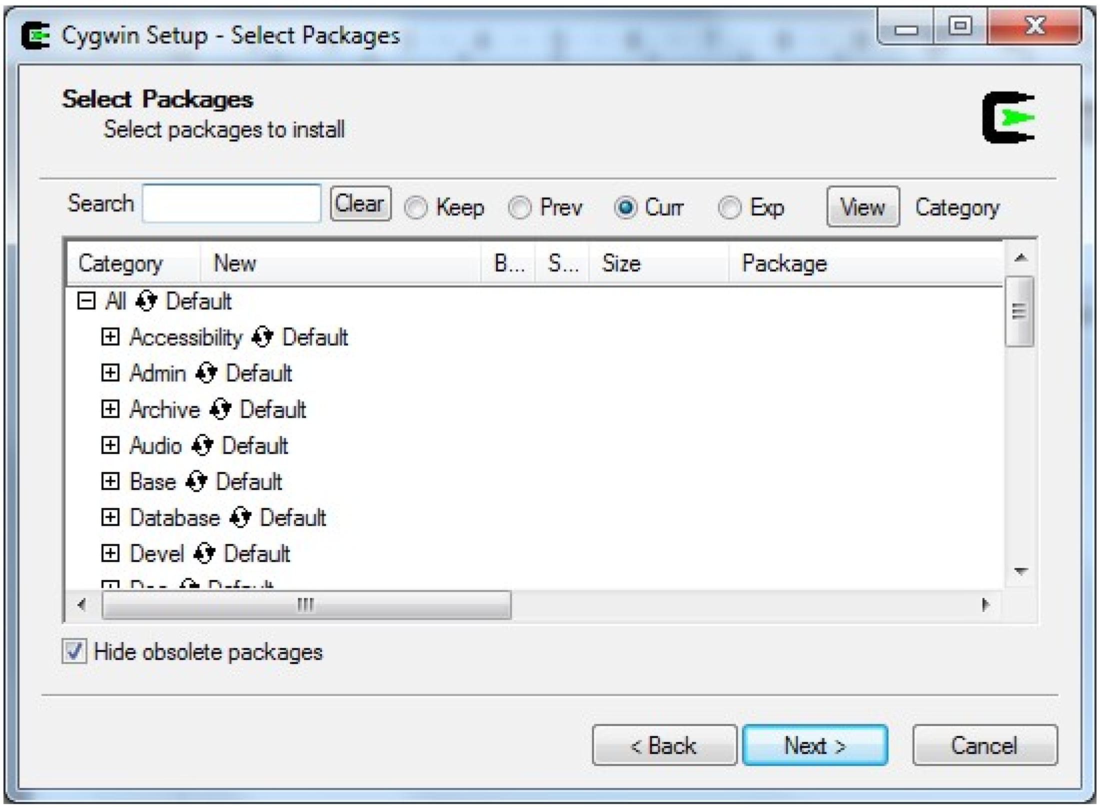
Select *Install from Internet*
Press *Next*

Enter *the location* and *the users* for Cygwin.
I entered **D:\cygwin** and selected the recommended user option.
Press *Next*.

Enter *the location* you want Cygwin to download its packages to
I entered **D:\cygwin\downloads**.
Press *Next*

I use a *Direct Connection* to the Internet, your connection settings may differ ...
Press *Next*

Choose *a mirror* near you.
Press *Next*.


Cygwin Setup - Select Packages

Select Packages
Select packages to install

Search [ ] Clear ○ Keep ○ Prev ● Curr ○ Exp View Category

Category	New	B...	S...	Size	Package
⊟ All ♻ Default					
⊞ Accessibility ♻ Default					
⊞ Admin ♻ Default					
⊞ Archive ♻ Default					
⊞ Audio ♻ Default					
⊞ Base ♻ Default					
⊞ Database ♻ Default					
⊞ Devel ♻ Default					

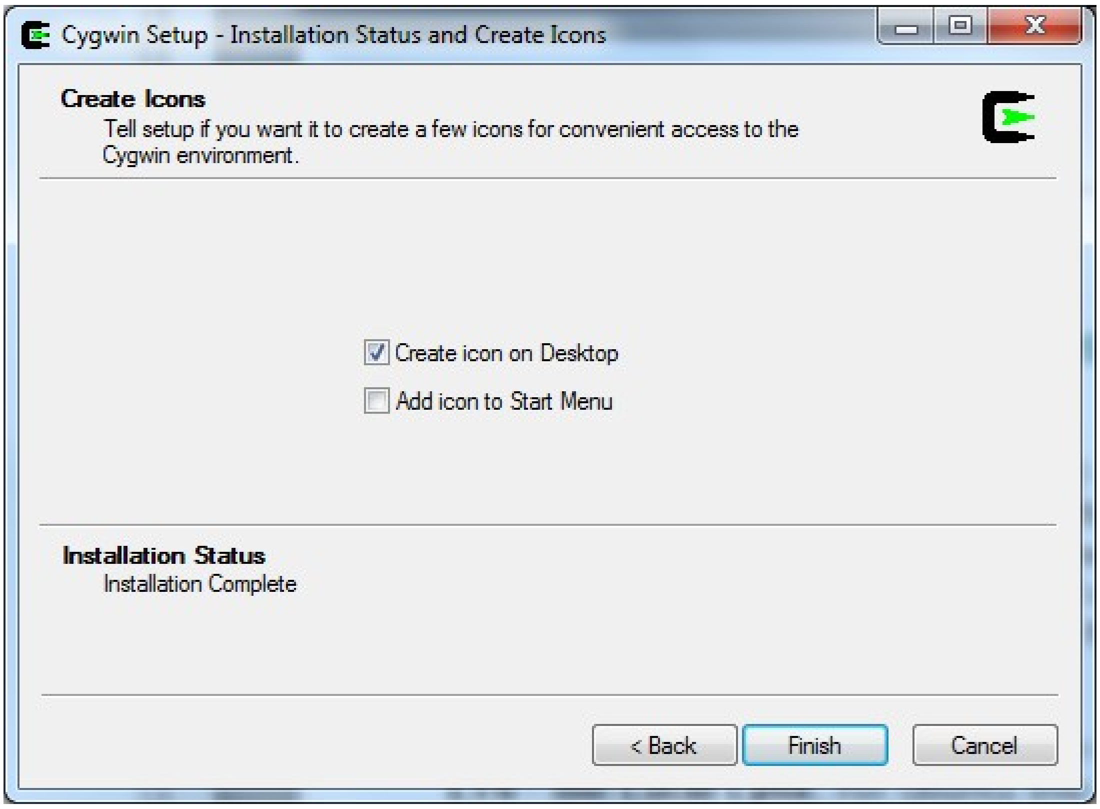
☑ Hide obsolete packages

< Back | Next > | Cancel

Finally we are getting to the packages (Linux utilities) that are going to be installed. For the most part the defaults are fine, but there are two packages that you want to select extra under the **Net**-heading (expand *that heading* and click on the *skip* in front of the packages ... the skip will be replaced by a version number) :
- openssh
- rsync

Press *Next*
Confirm that you want to select the packages that resolve the dependencies.
Press *Next*

The installation will now run for a bit ...



Create Icons
Tell setup if you want it to create a few icons for convenient access to the Cygwin environment.

- [✓] Create icon on Desktop
- [ ] Add icon to Start Menu

Installation Status
Installation Complete

Select how you want to be able to reach Cygwin.
Press *Finish*

Congratulations ! You are now the proud owner of a quite decent Linux environment on your Windows machine.

Non-root user – Ubuntu only

If the Linux machine for the Apposite Repository differs from the NetKernel machine, you will benefit from creating the same non-root user we created for the NetKernel machine.

```
your_user@ubuntumachine:~$ sudo groupadd --gid 1060 dexter
your_user@ubuntumachine:~$ sudo useradd --uid 1060 --gid 1060 -m \
 -d /home/dexter -s /bin/bash -c 'Apposite Repository' dexter
your_user@ubuntumachine:~$ sudo passwd dexter
```

Synchronization

Creating the repository – Windows 7

Create *a directory* to hold the repository:

```
C:\Users\your_user>mkdir d:\repo
```

Creating the repository – Ubuntu

Create a directory to hold the repository:

```
your_user@ubuntumachine:~$ sudo mkdir /repo
your_user@ubuntumachine:~$ sudo chown dexter:dexter /repo
```

Synchronizing the repository – Windows 7

Start your *Cygwin shell* (doubleclick the icon that was created on your desktop).

```
your_user@windowshost ~
$ mkdir /repo

your_user@windowshost ~
$ mount d:/repo /repo

your_user@windowshost ~
$ rsync -rv rsync://apposite.netkernel.org/download/repo/ /repo/
```

If you are using a firewall (you should) it will now ask you if **rsync** is allowed access to the Internet. Grant that access. A second later it will come back to ask if rsync may act as a server. It may.

The synchronization will take a while, the repository is (September 2010) about 250Mb.

```
...
packages/Y/
packages/Z/

sent 10335 bytes  received 265302027 bytes  304082.94 bytes/sec
total size is 265230673  speedup is 1.00
```

```
your_user@windowshost ~
$ umount /repo

your_user@windowshost ~
$ rmdir /repo

your_user@windowshost ~
$ exit
```

Synchronizing the repository – Ubuntu
Log on to the system as the non-root user we created earlier.

```
dexter@ubuntumachine:~$ rsync -rv \
rsync://apposite.netkernel.org/download/repo/ /repo/
```

The synchronization will take a while, the repository is (September 2010) about 250Mb.

```
...
packages/Y/
packages/Z/

sent 10335 bytes  received 265302027 bytes  318311.17 bytes/sec
total size is 265230673  speedup is 1.00
```

Verification – Windows 7
You should see two directories in the repository.

```
C:\Users\your_user>dir d:\repo
 Volume in drive D is xxxx
 Volume Serial Number is xxxx-xxxx

 Directory of d:\repo

09/09/2010  21:17    <DIR>          .
09/09/2010  21:17    <DIR>          ..
09/09/2010  21:17    <DIR>          netkernel
09/09/2010  21:17    <DIR>          packages
               0 File(s)              0 bytes
               4 Dir(s)  x bytes free
```

Verification – Ubuntu
You should see two directories in the repository.

```
dexter@ubuntumachine:~$ ls -la /repo
total 16
drwxr-xr-x  4 dexter dexter 4096 2010-09-09 21:31 .
drwxr-xr-x 23 root   root   4096 2010-09-09 21:06 ..
drwxr-xr-x  3 dexter dexter 4096 2010-09-09 21:31 netkernel
drwxr-xr-x 38 dexter dexter 4096 2010-09-09 21:31 packages
```

Use

There are several ways you can go about this. You can **serve** the Apposite Repository to the NetKernel instance(s) through a webserver. In Chapter 4 we do exactly that when I show how you can use NetKernel as a webserver.

Another option is to **map** the Apposite Repository over your internal network to the NetKernel instance(s). Windows has several options for just that and with **Samba**[21] you can easily map a Linux directory to a Windows machine (the other way around remains a tricky thing though).

For the rest of this Appendix I assume that you have **manually copied** the Apposite Repository to the machine that is running the NetKernel instance.

Whatever option you take, remember to frequently resynchronize with the central repository on the internet ! Once a month for example will not hurt at all.

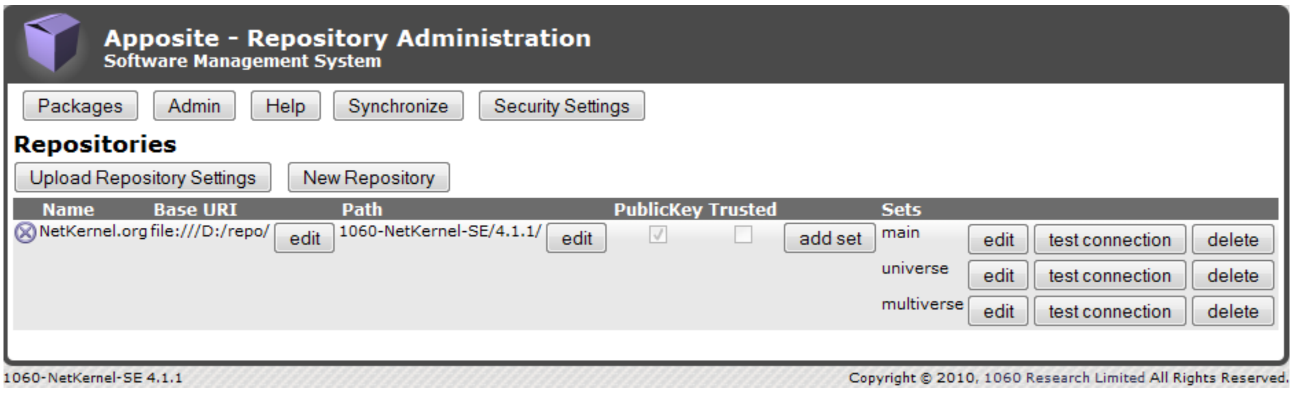
Automate this task or have it automated. It is all very well to be closed of from the *evil* internet, and no, you do not always need the latest and the greatest, but you do need security patches and the occasional new functionality. If you have to do it manually you'll forget after a while.

Activating your personal Apposite Repository – Windows 7

So, the assumptions are as follows :

- You are running *the NetKernel instance* on this machine.
- You've copied *the synchronized Apposite Repository* to this machine, in my case that is to **D:\repo**.

Navigate *your browser* to the NetKernel Apposite screen (http://localhost:1060 and so on, remember ?).
Press *the Admin button*.
Edit *the Base URI* of the repository so the screen looks like this :


Apposite - Repository Administration
Software Management System
Packages
Admin
Help
Synchronize
Security Settings

Repositories

Upload Repository Settings
New Repository

Name	Base URI		Path		PublicKey	Trusted		Sets			
NetKernel.org	file:///D:/repo/	edit	1060-NetKernel-SE/4.1.1/	edit	✓	☐	add set	main	edit	test connection	delete
								universe	edit	test connection	delete
								multiverse	edit	test connection	delete

1060-NetKernel-SE 4.1.1 Copyright © 2010, 1060 Research Limited All Rights Reserved.

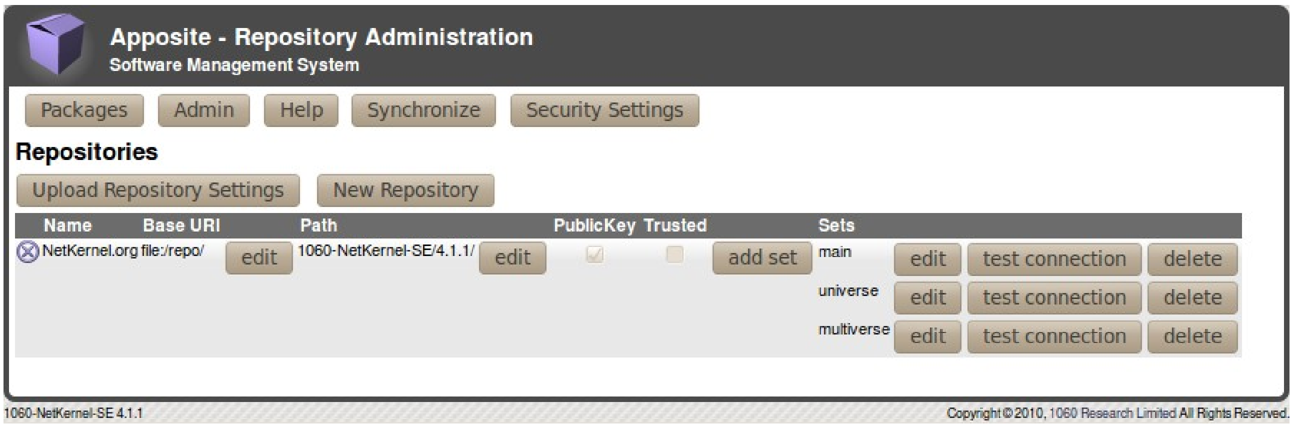
21 http://www.samba.org

Yes, the Base URI is now **file:///D:/repo/**. Windows loves slashes !
Make sure to test *the connection* !

Activating your personal Apposite Repository – Ubuntu
So, the assumptions are as follows :

- You are running *the NetKernel instance* on this machine.
- You've copied *the synchronized Apposite Repository* to this machine, in my case that is to **/repo**.

Navigate *your browser* to the NetKernel Apposite screen (http://localhost:1060 and so on, remember ?).
Press *the Admin button*.
Edit *the Base URI* of the repository so the screen looks like this :



Apposite - Repository Administration
Software Management System

Repositories

Name	Base URI		Path		PublicKey	Trusted		Sets			
NetKernel.org	file:/repo/	edit	1060-NetKernel-SE/4.1.1/	edit	✓	☐	add set	main	edit	test connection	delete
								universe	edit	test connection	delete
								multiverse	edit	test connection	delete

1060-NetKernel-SE 4.1.1
Copyright © 2010, 1060 Research Limited All Rights Reserved.

So, the Base URI is now **file:/repo/**.
Make sure to test *the connection* !

Conclusion

Setting up your own **Apposite Repository** is not hard at all. I would even dare to say it is an advisable thing to do :

- Your security team – if you have one – will be pleased.
- If you have multiple NetKernel instances running, your updates will be much swifter from a local (local as in 'on your local network') repository.

However, I would also advise to :

- Automate the synchronization with the official repository.
- Update your instances frequently.

One more thing to note ... the packages in a local repository are no less secure than those on the official repository. I quote :

> *The public apposite repository only has signed official releases of 1060 authorized packages. Both the individual packages and the complete repository metadata are signed. When you have a local copy inside your firewall the NKSE apposite client still performs full repository and package authentication and verification before permitting anything from the the mirror to be installed. So even though the library is local you can still treat it as the authentic trustworthy source of NKSE libraries and updates.*

And that's all I have to say about that.

Running NetKernel as a service / daemon started at boottime

Cover Story

For servers this is expected behavior, but imagine this : "You get up in the morning, you boot your desktop/laptop at home[22] and NetKernel is automatically started. Bliss.".

As you probably know, attaining the state of bliss is (in most religions) not one of the lighter matters. However, Peter Rodgers[23] has provided all that you require to run NetKernel as a service/daemon (on your server or on your home desktop) right here :
http://www.netkernel.org/forum/topic/649/1

If you feel comfortable with the instructions mentioned in that post, by all means follow them and ignore the rest of this Appendix. I'm going to do almost the same (you'll have to read further to see where I take a different approach ... muhahaha[24]).

Note that I only vouch for the platforms that I tested the procedures on. At this time those are **Windows 7** and **Ubuntu 10.04 LTS - the Lucid Lynx**. That means they will **probably** work on most current Windows platforms and Linux platforms. If you have been able to test on another platform (and want an honorable mention), let me know !

Preparation

Getting and installing YAJSW – Windows 7 only

No, I'm not using the Tanuki wrapper which you can find at http://wrapper.tanukisoftware.com and that is used in Peters post. Do not get me wrong, Tanuki is the authority on this task and their Community Edition is excellent. I was however very sorry to read this :

> *Release footnotes:*
> **1: 64-bit Windows versions of the Java Service Wrapper are not currently being made available in the Community Edition.*

22 If this is indeed the first thing you do after getting up (it is for me), get a life ... and tell me where you found one !
23 http://www.1060research.com/company/management/, Peter is the guy at *the top* ...
24 My 'evil laugh' imitation.

Very strange because the Linux 64-bit version is available. However, it does disqualify Tanuki for the time being. Somebody *used the force and read the* Tanuki-*source* however and the result (**Y**et **A**nother **J**ava **S**ervice **W**rapper) can be found at http://sourceforge.net/projects/yajsw/files/.

Download *the zip file* from there.
At this moment that is **yajsw-beta-10.3.zip**, your version may be different.
Unpack *the zip file* into a directory of your choice, in my case **D:\yajsw**.

Modifying YAJSW – Windows 7 only

In order to make YAJSW a bit more flexible we are going to alter the bat-files a bit. You can find those in the ... you guessed it ... bat-subdirectory, in my case **D:\yajsw\bat**.

Edit *setenv.bat*

```
...
rem configuration file used by all bat files
IF [%1]==[] (
set conf_file="%wrapper_home%/conf/wrapper.conf"
) ELSE (
set conf_file="%1"
)
...
```

Edit *installService.bat*

```
call setenv.bat %1
%wrapper_bat% -i %conf_file%
pause
```

Edit *startService.bat*

```
call setenv.bat %1
%wrapper_bat% -t %conf_file%
pause
```

Edit *stopService.bat*

```
call setenv.bat %1
%wrapper_bat% -p %conf_file%
pause
```

Edit *queryService.bat*

```
call setenv.bat %1
%wrapper_bat% -q %conf_file%
pause
```

Edit *uninstallService.bat*
```
call setenv.bat %1
%wrapper_bat% -r %conf_file%
pause
```

Getting the netkerneld script – Ubuntu only
You can use the location mentioned in the post for this, but actually you have the script on your system already. That is, if you kept the original jar-file you used to install NetKernel with.

Did you know that any jar-file – **1060-NetKernel-SE-4.1.1.jar** in my case – is actually a zip-file ? Rename *a copy of the .jar-file* to a *.zip-file* and unzip *it* (in a save location, you do not want to mess up your NetKernel install in this way).

The **netkerneld** script can be found in the bin-directory in that save location.

Use

Making the wrapper configuration for YAJSW – Windows 7 only
The YAJSW scripts expect a configuration file. You can put this configuration file (now that we've made the scripts somewhat more flexible) in NetKernels main etc-directory. In my case the file is **D:\NK4\etc\wrapper.netkernel.conf** and it contains the following (some lines are split due to the limitation of the linesize, but these are all single line definitions !) :

```
wrapper.java.command=java
wrapper.working.dir=D:\\\\NK4
wrapper.java.app.mainclass=BootLoader
wrapper.console.loglevel=INFO
wrapper.console.title=NetKernel
wrapper.ntservice.name=NetKernel
wrapper.ntservice.displayname=NetKernel
wrapper.ntservice.description=NetKernel
wrapper.java.classpath.1 =
D:\\\\NK4\\lib\\urn.com.ten60.core.boot-1.13.22.jar
wrapper.app.parameter.1 = D:\\\\NK4
wrapper.java.additional.1 = -Xmx128m
wrapper.java.additional.2 = -Xms128m
wrapper.java.additional.2 = -XX:SoftRefLRUPolicyMSPerMB=100
wrapper.java.additional.3 =
-Djava.protocol.handler.pkgs=org.ten60.netkernel.protocolhandler
wrapper.java.additional.4 =
-Dsun.net.client.defaultReadTimeout=20000
wrapper.java.additional.5 =
-Dsun.net.client.defaultConnectTimeout=20000
wrapper.logfile =
```

You should of course alter *this* according to your installation !

This is little more than you'll also find in the *normal* **netkernel.bat** startscript. If you're interested in knowing each and every possible parameter, check out : http://yajsw.sourceforge.net/YAJSW Configuration Parameters.html

I'm not quite happy with so many hardcoded parameters. I envision one extra step to generate this configuration file based on the available NK4 installation information.

Install and start the NetKernel service – Windows 7 only
You are now ready to install the service. There is however one more thing to take into account. Exactly that in fact. The scripts have to run with *Administrator* permissions. On Windows 7 there are several ways to accomplish this (I changed the property of the Command Prompt program to "Run As Administrator" ... now I just have to remember to turn it off again), just Google for a solution and you'll come up with one that suits you.

Nagivate to *the YAJSW bat-directory*

```
C:\Users\your_user>d:
D:\>cd yajsw\bat
D:\yajsw>
```

Install *the NetKernel service*

```
D:\yajsw\bat>installService.bat D:\NK4\etc\wrapper.netkernel.conf
```

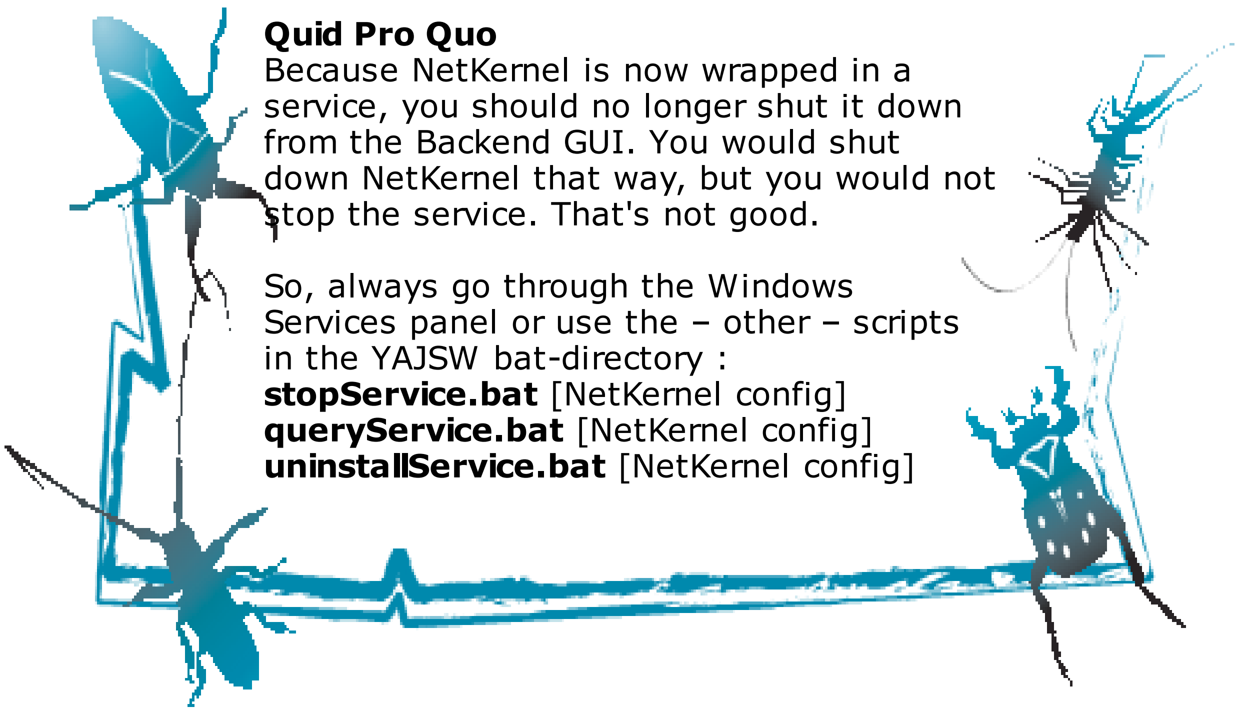
```
D:\yajsw\bat>call setenv.bat D:\NK4\etc\wrapper.netkernel.conf
"java" -Xmx30m -jar "d:\yajsw\bat\/../wrapper.jar" -i
"D:\NK4\etc\wrapper.netkernel.conf"
17-Sep-2010 15:26:21 org.apache.commons.vfs.VfsLog info
INFO: Using "C:\Users\YourUser\AppData\Local\Temp\vfs_cache" as
temporary files store.
Service NetKernel installed
```

That was easy, was it not ? You still have to start *the service* though ...

```
D:\yajsw\bat>startService.bat D:\NK4\etc\wrapper.netkernel.conf
```

```
D:\yajsw\bat>call setenv.bat D:\NK4\etc\wrapper.netkernel.conf
"java" -Xmx30m -jar "d:\yajsw\bat\/../wrapper.jar" -t
"D:\NK4\etc\wrapper.netkernel.conf"
17-Sep-2010 15:39:34 org.apache.commons.vfs.VfsLog info
INFO: Using "C:\Users\YourUser\AppData\Local\Temp\vfs_cache" as
temporary files store.
service started
```

And thats it. The NetKernel service will be restarted when you reboot your machine.



Quid Pro Quo
Because NetKernel is now wrapped in a service, you should no longer shut it down from the Backend GUI. You would shut down NetKernel that way, but you would not stop the service. That's not good.

So, always go through the Windows Services panel or use the – other – scripts in the YAJSW bat-directory :
stopService.bat [NetKernel config]
queryService.bat [NetKernel config]
uninstallService.bat [NetKernel config]

Modify and install the netkerneld script – Ubuntu only
Note

- You'll require superuser abilities to do the installation of the script.
- The wget-utility is required. Most Linux/Unix systems have this, if yours does not, install it first

Before we do anything else with the script, we're first going to make a few small adjustments.

Verify that *the parameters* at the top are correct

```
HOMEDIR=/usr/NK4
NK_USER=dexter
APPNAME=NetKernel

#User Editable variables
STARTGREPPID="ten60.pid=1"      #Must match the ten60.pid value
set in the netkernel.sh script
BACKENDPORT="1060"              #HTTP port of backend fulcrum
```

Comment out *this instruction* (just put a # in front of it) in the script

```
chown -R $NK_USER $HOMEDIR
```

It is an understandable but dangerous instruction. I personally prefer the daemon to fail at startup because a certain file *suddenly and magically* (yeah, right) has the wrong ownership. Maybe you prefer a flawless startup.

Now, we are ready to do the install. Move *the netkerneld script* from whereever you got it to /etc/init.d/ directory.

Change *the ownership* and *the permissions* of the script

```
your_user@ubuntumachine:~$ cd /etc/init.d
your_user@ubuntumachine:/etc/init.d$ sudo chown root:root
netkerneld
your_user@ubuntumachine:/etc/init.d$ sudo chmod 755 netkerneld
```

Verify if *the daemon* works

```
your_user@ubuntumachine:/etc/init.d$ sudo service netkerneld
Usage: /etc/init.d/netkerneld {start|stop|reboot|restart|status|
kill}
```

The next step is to link *the daemon* into the startup and shutdown of the machine. On Ubuntu this is very easy :

```
your_user@ubuntumachine:/etc/init.d$ sudo update-rc.d netkerneld
defaults
update-rc.d: warning: /etc/init.d/netkerneld missing LSB
information
update-rc.d: see <http://wiki.debian.org/LSBInitScripts>
 Adding system startup for /etc/init.d/netkerneld ...
   /etc/rc0.d/K20netkerneld -> ../init.d/netkerneld
   /etc/rc1.d/K20netkerneld -> ../init.d/netkerneld
   /etc/rc6.d/K20netkerneld -> ../init.d/netkerneld
   /etc/rc2.d/S20netkerneld -> ../init.d/netkerneld
   /etc/rc3.d/S20netkerneld -> ../init.d/netkerneld
   /etc/rc4.d/S20netkerneld -> ../init.d/netkerneld
   /etc/rc5.d/S20netkerneld -> ../init.d/netkerneld
```

You may safely ignore the LSB warning (unless you are a Debian purist, in which case you'll have rewritten the netkerneld script completely already).

Most Linux/Unix systems have these *runlevels*, so even if they do not have the **update-rc.d** tool, you can make the links yourself, for example :

```
your_user@ubuntumachine: $ ln -sf /etc/init.d/netkerneld
/etc/rc0.d/K20netkerneld
```

All that remains now is to start the daemon. You can do that by rebooting the machine or by starting *the service* yourself :

```
your_user@ubuntumachine:/etc/init.d$ sudo service netkerneld start
```

Verification – all environments
If all is well, you can now :
- fire up your favorite webbrowser
- enter http://localhost:1060

It pays to check that all works as expected. So do reboot your machines to see NetKernel rise and shine again !

APPENDIX D

Locking down your NetKernel instance

Prerequisites

<to be continued/>

Version Control

Raison d'être[25]

Statements
Backups protect you against *faulty hardware*.
Version control protects you against *faulty developers*.

Discussion
Most people will underscribe the two statements. We all know why the Challenger crashed[26], and we all know that the **I** in RA**I**D[27] stands for *Inexpensive*. And you might not believe this, but take the word of an ex-storage manager (me), the disks in your home pc are exactly the same ones that you can find in those very expensive storage-arrays from EMC[2], Hitachi, HP, IBM ... Exactly the same (even interchangeable if you replace the casing). Did you ever wonder why those arrays have upto 10 spare disks inside and why your company had to install a revolving door (and issue an access-at-all-times badge) for the storage technician replacing faulty disks ?

And yet, how many of you have a decent (regular, incremental) backup of their home pc ? It does – in this digital era - contain all photographs of your loved ones, your tax forms, your bank statements, your ...
Are those not important then ?

And yet, we all know that the better class of developer sometimes likes to ride his/her horse tangent of into some unknown direction. This is allowed (and often encouraged in the more successful companies, because although 9/10 it will be a useless exercise, the 1/10 pays the whole company for a year), but is it not important to be able to quickly revert back to the last working code ?

Conclusion
Backups and version control are necessary. This appendix will focus on version control (with Subversion) and explain how to make a safe backup of the version control directories. How frequently you make that backup and what you do with the resulting backupset is your choice. I hear people tell it is a choice between safety and paranoia. Tell that to the Challenger guys ...

25 Reason for existence
26 http://en.wikipedia.org/wiki/Space_Shuttle_Challenger_disaster
27 http://en.wikipedia.org/wiki/RAID

Preparation

Why Subversion[28] ?

I'm a Git[29] (man) myself and I've worked with CVS[30] and RCS[31] on Windows, HP/UX and Linux. A version control tool is however just that, a tool. *And a tool must fit the job at hand*. In the NetKernel News issue of September 24th 2010[32], Peter Rodgers explains why he feels *Subversion* fits the job for the 1060-team and why *Git* would probably not fit .

Besides being correct – insofar I can tell – that is couragous. Git is hot. Can you see the headline : "*Good enough for Linus Torvalds, but not for Peter Rodgers !?*" ? Luckily Linus would be the first to follow Peters reasoning.

So do I. And although Git would probably fit the – much simpler (as compared to the 1060-team system) – version control that I'm going to explain further on in this appendix, I'm going to use *Subversion* as well (and follow along in the structure Peter explains).

Installation

This is not an easy appendix and in order not to break the flow I'm not going to mix instructions for two operating systems. So I'm going to assume you use *Cygwin* on your Windows system and install *Subversion* there (it is under the **Devel**-heading). Appendix B explains how to setup *Cygwin*.

The only difference between the instructions will be the mounting and unmounting of the Windows drives on and off a Cygwin-directory. I'll mark these instructions as Cygwin-only.

If you have a Unix/Linux system it will probably have subversion installed already. If not you can find a package for your flavor at http://subversion.apache.org/packages.html.

For your understanding

- There is a book about *Subversion* available online at http://svnbook.red-bean.com. My goal is to walk you through some basics that will enable you to have source version control for NetKernel projects, not to better that book.
- I will describe a manual process. If (!) there is enough interest I will do an advanced add-on to this book and build a completely automated version control system/application for NKSE in there.
- I will work with a local repository. The *Subversion* book describes how to set up remote repositories.

28 http://subversion.apache.org//
29 http://git-scm.com/
30 http://www.nongnu.org/cvs/
31 http://www.cs.purdue.edu/homes/trinkle/RCS/
32 http://wiki.netkernel.org/wink/wiki/NetKernel/News/1/47/September_24th_2010

Basic Usage

At last. My apologies for the long-winded introduction to this Appendix. Version Control is one of those topics (*backup*, *disaster-recovery* and *security* are other examples) that many consider *just overhead*.

Setting up the repository

This is the heart of your system. Create *a location* for it. I'll go with **/svn** for the remainder of this Appendix (on Windows I created **d:\svn**, you'll see in a second how I map that).

Create *an empty repository* in that location.

```
youruser@yourmachine:~$ mkdir /svn                          # Cygwin only
youruser@yourmachine:~$ mount d:/svn /svn                   # Cygwin only

youruser@yourmachine:~$ svnadmin create /svn
```

Now, that was not hard, was it ?

Check *the repository*.

```
youruser@yourmachine:~$ ls /svn
README.txt  conf  db  format  hooks  locks
```

```
youruser@yourmachine:~$ cat /svn/README.txt
This is a Subversion repository; use the 'svnadmin' tool to
examine it.  Do not add, delete, or modify files here unless you
know how to avoid corrupting the repository.

Visit http://subversion.apache.org/ for more information.
```

Let me repeat that. Unless you know what you are doing, you should **not** (ever have to) touch the repository-directory or its contents !

```
umount /svn                                                 # Cygwin only
```

Layout module in the repository

Lets look at our *firstmodule* (from Chapter 2). The directory structure (in the repository - do not panic, I repeat - in the repository !) might look like this :

```
project-modules
  urn.org.netkernelbook.tutorial.firstmodule
    trunk
      src
    release
      1.0.0
      <other_release/>
    branch
      my_latest_experiment
      <other_branch/>
  <other_module/>
```

Underneath **trunk/src** we find the stuff we want every new release and every new branch to contain :

```
etc
  system
    SimpleDynamicImportHook.xml
resources
  dpml
  html
module.xml
```

Underneath **release** we will find all our versions. Each version starts out with what is in **trunk/src** and adds (and modifies) onto that :

```
1.0.0
  etc
    system
      SimpleDynamicImportHook.xml
  resources
    dpml
      hello.dpml
    html
      hello.html
  module.xml
```

```
1.0.1
  etc
    system
      SimpleDynamicImportHook.xml
  lib
    somelibrary.jar
  resources
    dpml
      hello.dpml  # Has been modified to use the library
    html
      hello.html
  module.xml      # Has been modified to use the library
```

A **branch** will be like a **release**, but is less official. Any experiment with the module can warrant a new branch.

```
my_briljant_experiment
    etc
    system
      SimpleDynamicImportHook.xml
  resources
    dpml
      briljant.dpml
    html
      briljant.html
  module.xml      # Has been modified to use the briljant dpml
```

So with *trunk*, *release* and *branch* you can cover most (if not all) of your needs. Note that a trunk can be updated with a the contents of a release or a branch if you feel like it, so changing the starting point for new releases/branches is possible (and I'll explain how to do that a bit further down).

Initial load of the repository
Create a *project-modules directory* somewhere. Do **not** use the project-modules directory underneath your NetKernel installation for this. I created one in my home directory :

```
youruser@yourmachine:~$ pwd
/home/youruser
```

Make sure *the repository* is available :

```
youruser@yourmachine:~$ mount d:/svn /svn            # Cygwin only
youruser@yourmachine:~$ ls /svn
README.txt  conf  db  format  hooks  locks
```

Import *project-modules* into the repository :

```
youruser@yourmachine:~$ svn import \
  /home/youruser/project-modules \
  file:///svn/project-modules -m "Initial Import"

Committed revision 1.
```

Congratulations, you've just started using Subversion !
Verify *the repository* :

```
youruser@yourmachine:~$ svnlook tree /svn
/
 project-modules/
```

Not a lot in there yet. But we'll soon change that !

```
youruser@yourmachine:~$ umount /svn                  # Cygwin only
```

You could now also remove the selfmade *project-modules* directory, but I suggest you keep it and use it to import modules (as I'll show in the next point).

Creating a module in the repository
Typically you do this **before** you create the module in NetKernel. Yes, I know we've already created our *firstmodule* (unless you started with all the Appendixes first ... not very likely). No worries, you'll be able to put the files in the repository !

But let us assume for a moment that we are just starting out with *firstmodule*. What we want to do then is create this structure in the repository :

```
project-modules
  urn.org.netkernelbook.tutorial.firstmodule
    trunk
      src
        etc
        resources
        module.xml
    release
    branch
```

Well, do so. **Not** in the repository and **not** in the NetKernel project-modules directory but in a place of your own choosing (I suggest underneath the selfmade project-modules) :

```
youruser@yourmachine:~$ mount d:/svn /svn          # Cygwin only
youruser@yourmachine:~$ pwd
/home/youruser
youruser@yourmachine:~$ cd project-modules
youruser@yourmachine:~$ mkdir \
 urn.org.netkernelbook.tutorial.firstmodule
youruser@yourmachine:~$ cd \
 urn.org.netkernelbook.tutorial.firstmodule
youruser@yourmachine:~$ mkdir -p trunk/src/etc/system \
 trunk/src/resources \
 release \
 branch
youruser@yourmachine:~$ touch \
 trunk/src/etc/system/SimpleDynamicImportHook.xml
youruser@yourmachine:~$ touch trunk/src/module.xml
```

If you want to, you may already fill *module.xml*. And *SimpleDynamicImportHook.xml*. Those two files are unlikely to change much in the lifetime of a module and thus belong completed in the trunk.

Import *urn.org.netkernelbook.tutorial.firstmodule* into the repository (command lines run until the \, they are **not** split at the – of project-modules, that's just the way the editor handles the limited linesize) :

```
youruser@yourmachine:~$ svn import \
  /home/youruser/project-
modules/urn.org.netkernelbook.tutorial.firstmodule \
  file:///svn/project-
modules/urn.org.netkernelbook.tutorial.firstmodule \
  -m "Create firstmodule"

Adding         /home/youruser/project-
modules/urn.org.netkernelbook.tutorial.firstmodule/release
Adding         /home/youruser/project-
modules/urn.org.netkernelbook.tutorial.firstmodule/trunk
Adding         /home/youruser/project-
modules/urn.org.netkernelbook.tutorial.firstmodule/trunk/src
Adding         /home/youruser/project-
modules/urn.org.netkernelbook.tutorial.firstmodule/trunk/src/modul
e.xml
Adding         /home/youruser/project-
modules/urn.org.netkernelbook.tutorial.firstmodule/trunk/src/resou
rces
Adding         /home/youruser/project-
modules/urn.org.netkernelbook.tutorial.firstmodule/trunk/src/etc
Adding         /home/youruser/project-
modules/urn.org.netkernelbook.tutorial.firstmodule/trunk/src/etc/s
ystem
Adding         /home/youruser/project
modules/urn.org.netkernelbook.tutorial.firstmodule/trunk/src/etc/s
ystem/SimpleDynamicImportHook.xml
Adding         /home/youruser/project
modules/urn.org.netkernelbook.tutorial.firstmodule/branch

Committed revision 2.
```

Verify *the repository* :

```
youruser@yourmachine:~$ svnlook tree /svn
/
 project-modules/
  urn.org.netkernelbook.tutorial.firstmodule/
   release/
   trunk/
    src/
     module.xml
     resources/
     etc/
      system/
       SimpleDynamicImportHook.xml
   branch/
```

Aha, now we are getting somewhere. And yes, I do know what remark you want to make by now (and if not, you will in a minute or so). Hold your water for bit, I'll answer before we are through with this appendix !

```
youruser@yourmachine:~$ umount /svn                    # Cygwin only
```

You may now safely remove the selfmade *firstmodule* directory. I suggest you keep the *project-modules* directory, so you have a location to create the repository-structure for future modules.

Creating a module-release in the repository

With the trunk nicely filled up it is time to start working on the first release of our module. This is a simple copy operation in the repository. I'm going to use the standard *major.minor.revision* numbering.

Make *a repository copy* of trunc/src to release/1.0.0

```
youruser@yourmachine:~$ mount d:/svn /svn              # Cygwin only

youruser@yourmachine:~$ svn copy \
 file:///svn/project-
modules/urn.org.netkernelbook.tutorial.firstmodule/trunk/src \
 file:///svn/project-
modules/urn.org.netkernelbook.tutorial.firstmodule/release/1.0.0
-m "Create release 1.0.0"

Committed revision 3.
```

Verify *the repository* :

```
youruser@yourmachine:~$ svnlook tree /svn
/
 project-modules/
  urn.org.netkernelbook.tutorial.firstmodule/
   release/
    1.0.0/
     module.xml
     resources/
     etc/
      system/
       SimpleDynamicImportHook.xml
   trunk/
    src/
     module.xml
     resources/
     etc/
      system/
       SimpleDynamicImportHook.xml
   branch/
```

That was easy was it not ? Yes, yes, I do know what you want to say. Let us clear the next step first and I'll answer.

```
youruser@yourmachine:~$ umount /svn                 # Cygwin only
```

Checking out a working-copy of the module-release
In this step we are actually going to do something in the NK4 project-modules directory, for that is where our working-copy is going to live. I'm going to assume your NK4 installation lives at **/usr/NK4**, alter the instructions accordingly if yours lives someplace else.

If you already created *firstmodule* in Chapter 2, move *the urn.org.netkernelbook.tutorial.firstmodule-1.0.0 directory* to a save location (away from /usr/NK4/project-modules) before you do the checkout. We'll reconciliate in a minute, but right now you want to save your work, for it is about to be overwritten.

Checkout *a working-copy* of release 1.0.0 from the repository.

```
youruser@yourmachine:~$ mkdir /usr/NK4              # Cygwin only
youruser@yourmachine:~$ mount d:/NK4 /usr/NK4       # Cygwin only
youruser@yourmachine:~$ mount d:/svn /svn           # Cygwin only
```

```
youruser@yourmachine:~$ svn checkout \
 file:///svn/project-
modules/urn.org.netkernelbook.tutorial.firstmodule/release/1.0.0 \
 /usr/NK4/project-
modules/urn.org.netkernelbook.tutorial.firstmodule-1.0.0

A    /usr/NK4/project-
modules/urn.org.netkernelbook.tutorial.firstmodule-
1.0.0/module.xml
A    /usr/NK4/project-
modules/urn.org.netkernelbook.tutorial.firstmodule-1.0.0/resources
A    /usr/NK4/project-
modules/urn.org.netkernelbook.tutorial.firstmodule-1.0.0/etc
A    /usr/NK4/project-
modules/urn.org.netkernelbook.tutorial.firstmodule-
1.0.0/etc/system
A    /usr/NK4/project-
modules/urn.org.netkernelbook.tutorial.firstmodule-
1.0.0/etc/system/SimpleDynamicImportHook.xml
Checked out revision 3.
```

You'll now notice that the *urn.org.netkernelbook.tutorial.firstmodule-1.0.0* directory is created in /usr/NK4/project-modules and filled with the contents of release 1.0.0 in the repository.

You'll also notice a *.svn* directory in *urn.org.netkernelbook.tutorial.firstmodule-1.0.0* and in every subdirectory. Leave those alone. They are used by *Subversion* to figure out your changes.

```
youruser@yourmachine:~$ umount /usr/NK4            # Cygwin only
youruser@yourmachine:~$ umount /svn                # Cygwin only
```

And here is the remark you've been wanting to make for some time now :
What is this sh*t ? All this extra work for a simple module like *firstmodule* and we haven't even reached the actual making (and keeping) of changes yet !

The answer to that is :
Yes.

The answer to that is also :
Yes. It does not get harder for a more complicated module though. This is as tough as it gets.

The answer to that is ... last but not least :
*Yes. I did tell you we were going to do this manually. This should be an automated process, integrated with the creation of a module. You should feel by now that this is not an easy matter. Decent version control requires planning (ahead). Now, go back to "**For your understanding**" and read the second point again.*

Making and keeping changes
Well, the time has come. If you saved the *firstmodule* you made in Chapter 2, copy the **content** of your saved *urn.org.netkernelbook.tutorial.firstmodule-1.0.0* directory to the directory with the same name under */usr/NK4/project-modules*. Just the content mind you ! You can now remove the saved version.

If you did not make the *firstmodule* yet, do so now. I'll wait.

If you did everything correctly the module should now work (again). It does ? Wonderful. Lets move on to keeping the changes in the Subversion repository as well.

Determine *the changes*.
```
youruser@yourmachine:~$ mount d:/NK4 /usr/NK4
youruser@yourmachine:~$ mount d:/svn /svn

youruser@yourmachine:~$ cd /usr/NK4/project-modules/
youruser@yourmachine:~$ cd \
 urn.org.netkernelbook.tutorial.firstmodule-1.0.0/

youruser@yourmachine:~$ svn status
M       module.xml
?       resources/html
?       resources/dpml
M       etc/system/SimpleDynamicImportHook.xml
```

You might think this is not correct, but lets examine what *Subversion* is telling us here :
1) File *module.xml* is modified. It is.
2) Directory *resources/html* is not in the tree of the repository. It isn't.
3) Directory *resources/dpml* is not in the tree of the repository. It isn't.
4) File *etc/system/SimpleDynamicImportHook.xml* is modified. It is.

Yes you say, fine, but I added a file in the html and another one in the dpml directory as well.

True, but you never told *Subversion* about the directories, so they are not (yet) in the repository tree and thus not taken into account. Let us remedy that first.

Tell *Subversion* about the new directories

```
youruser@yourmachine:~$ svn add resources/html
A         resources/html
A         resources/html/hello.html

youruser@yourmachine:~$ svn add resources/dpml
A         resources/dpml
A         resources/dpml/hello.dpml

youruser@yourmachine:~$ svn status
M       module.xml
A       resources/html
A       resources/html/hello.html
A       resources/dpml
A       resources/dpml/hello.dpml
M       etc/system/SimpleDynamicImportHook.xml
```

Looks a whole lot better, does it not ? Adding a directory to the tree automatically adds everything underneath it as well. So we do not have to add the new files to the tree manually.

Keep *the changes* (in the repository)

```
youruser@yourmachine:~$ svn commit -m "firstmodule completed"
Sending        etc/system/SimpleDynamicImportHook.xml
Sending        module.xml
Adding         resources/dpml
Adding         resources/dpml/hello.dpml
Adding         resources/html
Adding         resources/html/hello.html
Transmitting file data ...
Committed revision 4.
```

And that is all (ok, that was sarcasm). As long as you keep the repository save, you will always be able to checkout version 1.0.0 of *firstmodule* again.

Verify *Tom's claim*

```
youruser@yourmachine:~$ cd /usr/NK4/project-modules
youruser@yourmachine:~$ rm -rf \
 urn.org.netkernelbook.tutorial.firstmodule-1.0.0
```

```
youruser@yourmachine:~$ svn checkout \
file:///svn/project-
modules/urn.org.netkernelbook.tutorial.firstmodule/release/1.0.0 \
 /usr/NK4/project-
modules/urn.org.netkernelbook.tutorial.firstmodule-1.0.0

A    /usr/NK4/project-
modules/urn.org.netkernelbook.tutorial.firstmodule-
1.0.0/module.xml
A    /usr/NK4/project-
modules/urn.org.netkernelbook.tutorial.firstmodule-1.0.0/resources
A    /usr/NK4/project-
modules/urn.org.netkernelbook.tutorial.firstmodule-
1.0.0/resources/html
A    /usr/NK4/project-
modules/urn.org.netkernelbook.tutorial.firstmodule-
1.0.0/resources/html/hello.html
A    /usr/NK4/project-
modules/urn.org.netkernelbook.tutorial.firstmodule-
1.0.0/resources/dpml
A    /usr/NK4/project-
modules/urn.org.netkernelbook.tutorial.firstmodule-
1.0.0/resources/dpml/hello.dpml
A    /usr/NK4/project-
modules/urn.org.netkernelbook.tutorial.firstmodule-1.0.0/etc
A    /usr/NK4/project-
modules/urn.org.netkernelbook.tutorial.firstmodule-
1.0.0/etc/system
A    /usr/NK4/project-
modules/urn.org.netkernelbook.tutorial.firstmodule-
1.0.0/etc/system/SimpleDynamicImportHook.xml
Checked out revision 4.
```

Don't take my word for it (try it !), but although you intentionally removed its directory, *firstmodule* has been restored and will work correctly once more !

```
youruser@yourmachine:~$ umount /usr/NK4              # Cygwin only
youruser@yourmachine:~$ umount /svn                  # Cygwin only
```

So, to conclude this point here is a small overview:

1) *Subversion* has two types of changes

- file changes (modify file)
- tree changes (create directory/file, remove directory/file, move directory/file)

2) After making a change to your working copy you should

- Check your changes
- Inform the repository of any tree-changes.
- Commit or revert your changes

All is not well
You must be thinking I do pretty flawless development. Well, I do not. Actually I had to restart from scratch three times because I wanted every output I paste in this book to be **exactly** what you get on screen.

Since *Subversion* is pretty quick to create a new revision, that meant I had to restart every time I made a mistake. If anything this proves *Subversion* is pretty good at its job, keeping track of changes (regardless of whether they are the genuine thing or a mistake).

Restarting from scratch is of course not the way during normal development. Lets see what you can do when all is not well, we'll start with a small file change.

Check *the first three lines of module.xml*

```
youruser@yourmachine:~$ mount d:/NK4 /usr/NK4
youruser@yourmachine:~$ mount d:/svn /svn

youruser@yourmachine:~$ cd /usr/NK4/project-modules/
youruser@yourmachine:~$ cd \
 urn.org.netkernelbook.tutorial.firstmodule-1.0.0/

youruser@yourmachine:~$ svn status
youruser@yourmachine:~$ head --lines=3 module.xml
<module version="2.0">

  <meta>
```

Modify *module.xml*, by putting (with your text-editor) a comment in the second line.

Check *the first three lines of module.xml* again

```
youruser@yourmachine:~$ head --lines=3 module.xml
<module version="2.0">
  <!-- This is an intentional but useless comment              -->
  <meta>
```

<u>Determine</u> *your changes*

```
youruser@yourmachine:~$ svn status
M       module.xml

youruser@yourmachine:~$ svn diff
Index: module.xml
===================================================================
--- module.xml  (revision 4)
+++ module.xml  (working copy)
@@ -1,5 +1,5 @@
 <module version="2.0">
-
+  <!-- This is an intentional but useless comment             -->
   <meta>
     <identity>
       <uri>urn:org:netkernelbook:tutorial:firstmodule</uri>
```

<u>Revert</u> (undo) *the change* to module.xml

```
youruser@yourmachine:~$ svn revert module.xml
Reverted 'module.xml'
```

<u>Verify</u> *the revert* to module.xml

```
youruser@yourmachine:~$ svn diff
youruser@yourmachine:~$ svn status
youruser@yourmachine:~$ head --lines=3 module.xml
<module version="2.0">

  <meta>
```

```
youruser@yourmachine:~$ umount /usr/NK4             # Cygwin only
youruser@yourmachine:~$ umount /svn                 # Cygwin only
```

Next, we'll do (and undo) some tree-changes.
<u>Create</u> *a lib directory* for *firstmodule*

```
youruser@yourmachine:~$ mount d:/NK4 /usr/NK4
youruser@yourmachine:~$ mount d:/svn /svn

youruser@yourmachine:~$ cd /usr/NK4/project-modules/
youruser@yourmachine:~$ cd \
 urn.org.netkernelbook.tutorial.firstmodule-1.0.0/
youruser@yourmachine:~$ svn status
youruser@yourmachine:~$ mkdir lib
```

<u>Determine</u> *the change*

```
youruser@yourmachine:~$ svn status
?       lib
```

Add *the lib-directory* to the repository tree

```
youruser@yourmachine:~$ svn add lib
A          lib
```

Revert *the lib-directory*

```
youruser@yourmachine:~$ svn revert lib
```

Note that the lib-directory is still there ! We only reverted its addition to the repository tree.

Remove the *lib directory*

```
youruser@yourmachine:~$ rmdir lib
youruser@yourmachine:~$ svn status
```

```
youruser@yourmachine:~$ umount /usr/NK4                 # Cygwin only
youruser@yourmachine:~$ umount /svn                     # Cygwin only
```

Enough
That's enough *Subversion* basic use for this Appendix. Make sure to get the *Subversion* book and read at least Chapter 2 – Basic Usage. We'll finish this Appendix with some advanced stuff ...

Advanced Usage

There's a lot of topics that I could cover here. Again I'll point you to the *Subversion* book. There's just two things I still want to cover so you can safely go on your version control way without much further reading.

Integrating a release/branch into the trunk
Release 1.0.0 of our *firstmodule* is performing perfectly and you now want all of it in the *trunk* so future releases and branches can work with the perfect code.

Create *a working copy of the trunk*

```
youruser@yourmachine:~$ mount d:/NK4 /usr/NK4
youruser@yourmachine:~$ mount d:/svn /svn
```

```
youruser@yourmachine:~$ svn checkout \
 file:///svn/project-
modules/urn.org.netkernelbook.tutorial.firstmodule/trunk/src \
 /home/youruser/trunk-working-copy
A    /home/youruser/trunk-working-copy/module.xml
A    /home/youruser/trunk-working-copy/resources
A    /home/youruser/trunk-working-copy/etc
A    /home/youruser trunk-working-copy/etc/system
A    /home/youruser/trunk-working-
copy/etc/system/SimpleDynamicImportHook.xml
Checked out revision 4.
```

Verify *release 1.0.0*.

```
youruser@yourmachine:~$ cd /usr/NK4/project-modules/
youruser@yourmachine:~$ cd \
 urn.org.netkernelbook.tutorial.firstmodule-1.0.0/
youruser@yourmachine:~$ svn status
youruser@yourmachine:~$ svn update
At revision 4.
```

Merge *release 1.0.0* into the working copy of the trunk.

```
youruser@yourmachine:~$ cd /home/youruser/trunk-working-copy
youruser@yourmachine:~$ svn merge –reintegrate \
 file:///svn/project-
modules/urn.org.netkernelbook.tutorial.firstmodule/release/1.0.0
--- Merging differences between repository URLs into '.':
U    module.xml
A    resources/html
A    resources/html/hello.html
A    resources/dpml
A    resources/dpml/hello.dpml
U    etc/system/SimpleDynamicImportHook.xml

youruser@yourmachine:~$ svn commit \
 -m "Merge release 1.0.0 into trunk"
Sending        .
Sending        etc/system/SimpleDynamicImportHook.xml
Sending        module.xml
Adding         resources/dpml
Adding         resources/dpml/hello.dpml
Adding         resources/html
Adding         resources/html/hello.html
Transmitting file data ..
Committed revision 5.

youruser@yourmachine:~$ cd ..
youruser@yourmachine:~$ rm -rf trunk-working-copy
```

You could now actually remove the release 1.0.0 from the repository (with an *svn delete*), since it is now contained within the trunk. For a branch this would be a logical thing to do, a release number retains its validity and since Subversion is very storage efficient we'll leave it as it is.

```
youruser@yourmachine:~$ umount /usr/NK4              # Cygwin only
youruser@yourmachine:~$ umount /svn                  # Cygwin only
```

Taking a backup of your repository
Is actually a very easy task.

Make *hotcopy* of repository.

```
youruser@yourmachine:~$ mount d:/svn /svn
youruser@yourmachine:~$ svnadmin hotcopy /svn
/home/tomgeudens/svn-backup
youruser@yourmachine:~$ svnadmin verify /home/tomgeudens/svn-
backup
* Verified revision 0.
* Verified revision 1.
* Verified revision 2.
* Verified revision 3.
* Verified revision 4.
* Verified revision 5.
youruser@yourmachine:~$ umount /svn                    # Cygwin only
```

There, the *svn-backup* directory is a complete and valid copy of your repository (and can simply be put in the place of the original if you need to restore). As I said earlier, I leave it up to you to do something with it.

Not easy

This was not an easy appendix. Not for me to write and probably (despite my best efforts) not for you to read and follow. The reason is that although version control is very necessary, it also should be automated as much as possible.

If you have version control in place already for your development, use it. If you do not I hope this appendix gave you some ideas on how it can be done.