Miscellaneous Issues Regarding the reposTools Package

Jeff Gentry

January 23, 2004

1 Overview

This article will go over some miscellaneous issues that are handled by the *reposTools* package that go beyond the direct handling of repositories. These were things that were necessary to complete the *reposTools* code but were either unimplemented currently in R or we felt that an improvement could be made to the existing implementation.

For the purposes of demonstrating some functionality, we will be loading the *reposTools* package.

```
> library(reposTools)
```

2 Version Numbers

Currently in R, version numbers are handled as character strings and can be compared using the function `compareVersion`. As the *reposTools* package uses version numbers heavily, we felt that it would be better to provide a richer set of operations for versions. To do this, a `S4` class was constructed, `VersionNumber`. It allows for standard comparison operators (e.g. `==`, `>`, `<=`, etc), max/min finding, etc.

A constructor function has been provided to simplify the process of creating a `VersionNumber` object. The function `buildVersionNumber` is passed a string representing a version number and will output the created `VersionNumber` object.

```
> a <- buildVersionNumber("1.2-3")
> a

[1] "1.2-3"
```

A note about the format of version numbers. In the `VersionNumber` class, valid separator characters between version subsections are `.` and `-`. It will take the first section and treat it as the major version, the second subsection as the

minor version and any remaining subsections as the revision. There must be at least a major version, but anything else is optional.

```
> a <- buildVersionNumber("1")
> a

[1] "1"

> b <- buildVersionNumber("1.2")
> b

[1] "1.2"

> c <- buildVersionNumber("1.2.3")
> c

[1] "1.2.3"

> d <- buildVersionNumber("1-2.3-4.5.6-7")
> d

[1] "1-2.3-4.5.6-7"
```

Also as noted, `VersionNumber` objects can be compared using standard operators:

```
> a

[1] "1"

> b

[1] "1.2"

> a > b

[1] FALSE

> a < b

[1] TRUE

> a == a

[1] TRUE

> a != b

[1] TRUE

> max(a, b)

[1] "1.2"

> min(a, b)

[1] "1"
```

3 Package Information

Quite often it is useful to tie a version number to a package name. It is likely that one won't just need a particular package, but also a particular version of that package. To facilitate this, we've provided a small class `pkgInfo`. As with `VersionNumber` it provides a constructor function `buildPkgInfo`.

```
> b

[1] "1.2"

> z <- buildPkgInfo(name = "Biobase", vers = b)
> z

[1] "Biobase: 1.2"
```

4 Dependencies

An important aspect to maintaining a user's R library is insuring that for any given package that any packages it depends on are also installed and that the user has the proper R version installed. Currently, the standard for writing R packages declares that in the DESCRIPTION file, any package dependencies should be noted with the `Depends:` field. We've first constructed a system that provides for more granularity regarding the severity of a dependency. Next, we've provided a set of tools that allow the user greater control over handling, resolving and dealing with dependency issues.

We have defined three levels of dependencies:

depends The item directly depends on the package specified, and will be considered broken if it is not installed.

suggests The item uses the specified package, but only in tangential ways. A user will generally be able to achieve normal functionality without it installed.

uses The item is only using the specified package in examples, and the user will have full access to all functionality without it installed.

These three types of dependencies are used for both packages and vignette files. For packages, one can label them in much the same manner as previously (putting in 'depends', 'suggests' and 'uses' fields into the DESCRIPTION file of a package) and for vignettes one should use the header fields `%VignetteDepends{<package1>,<package2>}, %VignetteSuggests{} and %VignetteUses{}`. These dependencies are listed exactly like the current syntax (A comma separated list in the format <name> followed by optional version requirements, e.g. (>= 1.0)).

There are provided a pair of functions that allow a user to directly interact with dependency issues. The first is `unresolved.depends`. This function will

take a package name, and will return a listing of what packages are not currently installed for all three levels of dependencies:

```
> unresolved.depends("Biobase")

list()
```

Alternatively, one can use the `load.depends` function. This is similar to `unresolved.depends`, except that it will attempt to load any dependencies and return with an error if any are not installed.

```
> load.depends("Biobase")
```

Both of these functions have a similar signature: load.depends(x,suggests=TRUE,uses=TRUE,Rversion=TR

Also, in both cases, an output of `NULL` implies that there are either no unresolved dependencies or that there are no dependencies to load, respectively.

An explanation of the parameters:

x The package or vignette to check

suggests If TRUE, the system will check/load `suggests` level dependencies.

uses If TRUE, the system will check/load `uses` level dependencies.

Rversion If TRUE, the system will check any listed dependencies on the R version to the user's version of R.

A similar function is `resolve.depends`. This isn't primarily intended for end users (although there's no reason why one can't call it directly) but rather mainly used by the *reposTools* install/update/remove suite of commands. This function is given a `pkgInfo` object, and will see if it would break any dependencies from currently installed packages to install (or remove) this package. For instance, if package X depends on package Y and the user wishes to remove package Y, this would be a situation detected by `resolve.depends`. For instance, if we wish to remove package *Biobase* from our system:

```
> z

[1] "Biobase: 1.2"

resolve.depends(z,remove=TRUE)

Error in resolve.depends(z, remove = TRUE) :
        Can not continue:
affy version 1.2.1 depends on Biobase
annotate version 1.0 depends on Biobase
```

Or to view this in a more normal usage:

```
remove.packages2("Biobase")
```

```
Note: argument `lib' is missing: using /home/jgentry/R/library
Error in resolve.depends(buildPkgInfo(pkg), force, remove = TRUE) :
        Can not continue:
affy version 1.2.1 depends on Biobase
annotate version 1.0 depends on Biobase
```

As you can see, the `remove.packages2` function will not allow us to remove *Biobase* from our system. These functions do allow a parameter `force` (default is FALSE), which if TRUE will ignore any dependencies.