rBiopaxParser Short Vignette

12th July 2012
Frank Kramer <dev@frankkramer.de>

The aim of this document is to help the user get accustomed with the package and to provide a step-by-step introduction on how to get started. For a deeper understanding of how Biopax instances are composed, it is strongly encouraged to take a look at the Biopax definition, especially the class inheritance tree and the list of properties for each class is very helpful. The language definition as well as further information can be found at http://www.biopax.org .

This vignette contains a installation instructions as well as a quick listing of working code to get started with the package right away.

Index

0 **BioPAX**

A plethora of databases offer a vast knowledge about biological signaling. Biopax is implemented in the Web Ontology Language OWL, an XML-based markup language. It allows the users to store and exchange pathway knowledge in a well-documented and standardized way. In simplified terms one can say, that the main class, the pathway, is build up from a list of interactions. Interactions themselves provide a link from one controlling molecule to a controlled molecule. Molecule instances, including their properties like names, sequences or external references are defined within the Biopax model. This package will hopefully ease the task of working with BioPAX data within R.

1 **Installing the package**

1.1 Prerequisites

This package suggests package RCurl to download Biopax files from the web. This package needs package XML to parse the Biopax .owl files. This package needs package graph to build graphs/networks from the data. This package suggests package Rgraphviz to visualize networks. To install directly from github you need package devtools.

Installation or running certain functions MIGHT fail if these are not met. Please read through the following instructions.

1.1.1 Prerequisites for Linux users

XML:
Make sure your linux has library libxml2 installed. This is almost always the case. Otherwise run in your shell
```
sudo apt-get install libxml2
```
will fix this issue. You will now be able to install R package XML, this should be automatically done when you install rBiopaxParser, or you can run within R:
```
install.packages("XML")
```

RCurl:
RCurl is only needed for a convenience function to download Biopax files directly within R. You can skip this step if you already have the Biopax data downloaded.
Make sure your linux has library libcurl installed and curl-config in your path.

Check out
```
locate libcurl
locate curl-config
```
If these are not found (usually the developer version is missing), most Linux users can usually fix this by running
```
sudo apt-get install libcurl4-openssl-dev
```
You will now be able to install R package RCurl, this should be automatically done when you install rBiopaxParser, or you can run within R:
```
install.packages("RCurl")
```
If you encounter more problems check out
http://www.omegahat.org/RCurl/FAQ.html

graph:
Package graph has moved from CRAN to Bioconductor recently, you might encounter an error saying that package graph is not available for your distribution when calling install.packages("graph").
Check out http://bioconductor.org/packages/release/bioc/html/graph.html or call
```
source("http://bioconductor.org/biocLite.R")
biocLite("graph")
```
to install it.

Rgraphviz:
Rgraphviz is used to layout the graphs generated in this package. You can layout and plot these yourself if you want to.
Make sure your linux has package graphviz installed.
If this is not the case, many linux users can usually fix this by running
sudo apt-get install graphviz
You will now be able to install R package Rgraphviz using:
source("http://bioconductor.org/biocLite.R")
biocLite("Rgraphviz")
If you encounter more problems check out
http://www.bioconductor.org/packages/release/bioc/html/Rgraphviz.html

devtools:
Package devtools is available at CRAN. Run
```
install.packages("devtools")
```
to install it.

1.1.2 Prerequisites for Windows users

XML & RCurl:
These packages depend on linux libraries, however Brian Ripley has put some work into this to enable Windows users.

Check out http://www.stats.ox.ac.uk/pub/RWin/bin/windows/contrib/ for these two packages for your R version.
Download first XML...zip and then RCurl...zip and install them locally on your machine.

graph:

Package graph has moved from CRAN to Bioconductor recently, you might encounter an error saying that package graph is not available for your distribution when calling install.packages("graph").
Check out http://bioconductor.org/packages/release/bioc/html/graph.html or run

```
source("http://bioconductor.org/biocLite.R")
biocLite("graph")
```

to install it.

Rgraphviz:

Rgraphviz is used to layout the graphs generated in this package. You can layout and plot these yourself if you want to.
Make sure your machine has Graphviz installed, it can be found at:
http://www.graphviz.org
Click on Download -> Windows. After installing graphviz you will now be able to install R package Rgraphviz using:

```
source("http://bioconductor.org/biocLite.R")
biocLite("Rgraphviz")
```

If you encounter more problems check out
http://www.bioconductor.org/packages/release/bioc/html/Rgraphviz.html

devtools:

Package devtools is available at CRAN. For Windows this seems to depend on having Rtools for Windows installed. You can download and install this from:
http://cran.r-project.org/bin/windows/Rtools/
To install R package devtools call

```
install.packages("devtools")
```

1.2 Installation

If everything went well you will be able to install the rBiopaxParser package from:
https://github.com/frankkramer/rBiopaxParser
(click the ZIP button to download it)

OR install directly using the devtools package:

```
library(devtools)
install_github(repo="rBiopaxParser", username="frankkramer")
```

2 Download Biopax Data

Many online databases of pathway knowledge offer an export in Biopax format. This package gives the user a shortcut to download Biopax exports directly from database providers from the web. A list of links to commonly used databases is stored internally and the user can select from which source and which export to download. The data is stored in the working directory. If needed the downloaded data is unzipped.
Currently only the NCI website with exports of the Pathway Interaction Database (PID), BioCarta and Reactome exports are linked.
The following command downloads the BioCarta export from the NCI website.

```
downloadBiopaxData("NCI","biocarta")
```

After the download is finished the on-screen output informs the user of success and name of the downloaded file.

3 Parse Biopax Data

Biopax data can be parsed into R using the rBiopaxParser. This function reads in a Biopax .owl file and generates the internal data.frame format used in this package. This function can take a while with really big Biopax files like NCIs Pathway Interaction Database or Reactome.

The following command reads in the Biopax file which was previously downloaded into variable biopax.

```
biopax = readBiopax("BioCarta.bp2.owl")
```

A summary is automatically generated and can be accessed calling

```
biopax$summary
```

4 Internal Data Model

5 Accessing the internal representation

Many convenience functions are available that will aid the user in selecting certain parts or instances of the biopax model. Generally these functions will always require the parsed biopax model as parameter as well as other parameters that differ from function to function.

The most basic functions to select distinct instance are getBiopaxInstancesByID, getBiopaxInstancesByName, getBiopaxInstancesByType. All of these do exactly as the name suggest and are also vectorized to allow the user to select multiple instances.
These functions return data.frames according to the internal data model.

The next type of selecting functions are (a) getBiopaxInstancesList, (b) getPathwayList, (c) getPathwayComponentList which return a vector of IDs and names of either all instances, all pathways or of all pathway components respectively.

The functions getPathways and getPathway returns all or a a distinctive pathway of a biopax model, whereas getPathwayComponents returns all pathway component ids of a pathway.

The functions getReferencedIDs and getReferencedInstances, which can optionally both be called recursively, are passed a biopax model and an instance ID.
The return value is either the IDs or the complete instances of all instances that are referenced by the instance supplied.

```
pw_list = getPathwayList(biopax)
pw_complete = getPathways(biopax)

pwid1 = "pid_p_100002_wntpathway"    #wnt
pwid2 = "pid_p_100146_hespathway"    #segmentation clock
pwid3 = "pid_p_100074_pitx2pathway"   #pitx2

pw_1  = getPathway(biopax,pwid1)
pw_1_component_list = getPathwayComponentList(biopax,pwid1)
pw_1_components = getPathwayComponents(biopax,pwid1)
pw_2  = getPathway(biopax,pwid2)
pw_2_component_list = getPathwayComponentList(biopax,pwid2)
pw_2_components = getPathwayComponents(biopax,pwid2)
pw_3  = getPathway(biopax,pwid3)
pw_3_component_list = getPathwayComponentList(biopax,pwid3)
pw_3_components = getPathwayComponents(biopax,pwid3)
```

6 Visualization

These functions transform Biopax pathways into regulatory graphs. However, there are some caveats.

Transform into a regulatory graph:

```
pw_1_adj   = pathway2AdjacancyMatrix(biopax, pwid1, expandSubpathways=TRUE,
      splitComplexMolecules=TRUE, verbose=TRUE)
pw_1_graph = pathway2RegulatoryGraph(biopax, pwid1,
      splitComplexMolecules=TRUE, verbose=TRUE)
pw_2_adj   = pathway2AdjacancyMatrix(biopax, pwid2, expandSubpathways=TRUE,
      splitComplexMolecules=TRUE, verbose=TRUE)
pw_2_graph = pathway2RegulatoryGraph(biopax, pwid2,
      splitComplexMolecules=TRUE, verbose=TRUE)
pw_3_adj   = pathway2AdjacancyMatrix(biopax, pwid3, expandSubpathways=TRUE,
      splitComplexMolecules=TRUE, verbose=TRUE)
pw_3_graph = pathway2RegulatoryGraph(biopax, pwid3,
      splitComplexMolecules=TRUE, verbose=TRUE)
```

Layout the graphs using Rgraphviz:

```
pw_1_graph_laidout = layoutRegulatoryGraph(pw_1_graph)
pw_2_graph_laidout = layoutRegulatoryGraph(pw_2_graph)
pw_3_graph_laidout = layoutRegulatoryGraph(pw_3_graph)
```

Plot the graphs:

```
plotRegulatoryGraph(pw_1_graph)
plotRegulatoryGraph(pw_2_graph)
plotRegulatoryGraph(pw_3_graph)
```

Merge graphs and render them (this time disable re-layouting for the plot function:

```
merged_graph = uniteGraphs(pw_1_graph_laidout,pw_2_graph_laidout)
plotRegulatoryGraph(merged_graph, layoutGraph=FALSE)
```

7 Modifying Biopax

Instead of merging the regulatory graph representations it is also possible to merge the biopax pathways directly and add this new, merged pathway directly into the biopax model.

```
biopax = mergePathways(biopax, pwid1, pwid2, NAME="mynewpw1",
     ID="mynewpwid1")
mergedpw_graph = pathway2RegulatoryGraph(biopax, "mynewpwid1",
     splitComplexMolecules=TRUE, verbose=TRUE)
plotRegulatoryGraph(layoutRegulatoryGraph(mergedpw_graph))
```

Although it is possible to directly edit the parsed Biopax data by accessing biopax$df, there are quite a few convenience functions to make life easier. In the following code block functions addPhysicalEntity, addPhysicalEntityParticipant, addBiochemicalReaction, addControl and addPathway will be used to build 2 pathways with 2 controls between 3 proteins each.

Start out with adding 5 proteins, their corresponding physicalEntityParticipant instances and a biochemical reaction where they do something to themselves.

```
temp = biopax
for(i in LETTERS[1:5]) {
        temp = addPhysicalEntity(temp, class="protein",
          NAME=paste("myprot",i,sep="_"), ID=paste("myproteinid",i,sep="_"))
        temp = addPhysicalEntityParticipant(temp,
          referencedPhysicalEntityID=paste("myproteinid",i,sep="_"),
          ID=paste("myPEPid",i,sep="_"))
        temp = addBiochemicalReaction(temp, LEFT=paste("myPEPid",i,sep="_"),
          RIGHT=paste("myPEPid",i,sep="_"), ID=paste("myBCRid",i,sep="_"))}
```

Now we add some controls between those proteins.

```
temp = addControl(temp, CONTROL_TYPE="ACTIVATION",
CONTROLLER="myPEPid_A", CONTROLLED=c("myBCRid_B"),
ID="mycontrol_1")
temp = addControl(temp, CONTROL_TYPE="INHIBITION",
CONTROLLER="myPEPid_A", CONTROLLED=c("myBCRid_C"),
ID="mycontrol_2")
temp = addControl(temp, CONTROL_TYPE="ACTIVATION",
CONTROLLER="myPEPid_C", CONTROLLED=c("myBCRid_D"),
ID="mycontrol_3")
temp = addControl(temp, CONTROL_TYPE="INHIBITION",
CONTROLLER="myPEPid_C", CONTROLLED=c("myBCRid_E"),
ID="mycontrol_4")
```

These interactions will be used as pathway components for new pathways.

```
temp = addPathway(temp, NAME="mynewpw2",
        PATHWAY_COMPONENTS=c("mycontrol_1","mycontrol_2"),
        ID="mynewpwid2")
temp = addPathway(temp, NAME="mynewpw3",
```

```
        PATHWAY_COMPONENTS=c("mycontrol_3","mycontrol_4"),
        ID="mynewpwid3")
temp = mergePathways(temp, "mynewpwid2", "mynewpwid3", NAME="mynewpw4",
        ID="mynewpwid4")
```

Now these new pathways are ready to be viewed!

```
mynewpw2_graph = pathway2RegulatoryGraph(temp, "mynewpwid2",
        splitComplexMolecules=TRUE, verbose=TRUE)
mynewpw3_graph = pathway2RegulatoryGraph(temp, "mynewpwid3",
        splitComplexMolecules=TRUE, verbose=TRUE)
mynewpw4_graph = pathway2RegulatoryGraph(temp, "mynewpwid4",
        splitComplexMolecules=TRUE, verbose=TRUE)

plotRegulatoryGraph(layoutRegulatoryGraph(mynewpw2_graph))
plotRegulatoryGraph(layoutRegulatoryGraph(mynewpw3_graph))
plotRegulatoryGraph(layoutRegulatoryGraph(mynewpw4_graph))
```

Finally properties as well as complete instances can be removed from the current
Biopax model by calling:

```
temp = removeProperties(temp,instanceid="mynewpwid2",
        properties="PATHWAY-COMPONENTS")
temp = removeInstance(temp,instanceid="mynewpwid3")
```

8 Writing out in Biopax Format

Writing out an internal Biopax model into a valid .owl file is very easy.
Simply call:

```
writeBiopax(biopax, file="test.writeBiopax.owl")
```