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 provide a step-by-step introduction on how to get started. For a deeper understand of how Biopax instances are composed, it is strongly encouraged to take a look at the Biopax definition, especially the class inheritance tree and which properties each class has is very helpful. The language definition as well as further information can be found at http://www.biopax.org .

This is a quick listing of working code to get started with the package right away. A longer and more detail vignette will follow!

0. Biopax

A plethora of databases offer a vast knowledge about biological signaling 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 are a linkage of a controlling molecule and a controlled molecule.

To get to things right away, you can download this package at:
https://github.com/frankkramer/rBiopaxParser

OR install directly with the devtools package:

```
install.packages(“devtools”)
library(devtools)
install_github(repo="rBiopaxParser", username="frankkramer")
```

1. Retrieve 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.

2. 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
```

3. Internal Data Model

4. Selecting and Viewing from 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)
```

5. Visualization

If you can't see it, it's not there!
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 (layouting them again would break the node colors):

```
merged_graph = uniteGraphs(pw_1_graph_laidout,pw_2_graph_laidout)
Rgraphviz::renderGraph(merged_graph)
```

6. Modifying Biopax

7. 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")
```