flowWorkspace: A Package for Importing flowJo Workspaces into R

Greg Finak <gfinak@fhcrc.org>

January 25, 2012

1 Purpose

The purpose of this package is to provide functionality to import relatively simple *flowJo* workspaces into R. By this we mean, accessing the samples, groups, transformations, compensation matrices, gates, and population statistics in the *flowJo* workspace, and replicating these using (primarily) *flowCore* functionality.

2 Why Another flowJo Workspace Import Package?

There was a need to import *flowJo* workspaces into R for comparative gating. The *flowFlowJo* package did not meet our needs. Many groups have legacy data with associated flowJo XML workspace files in version 2.0 format that they would like to access using BioConductor's tools. Hopefully this package will fill that need.

3 Support

This package supports importing of **Version 2.0 XML workspaces only**. We cannot import **.jo** files directly. You will have to save them in XML workspace format, and ensure that that format is *workspace version 2.0*. The package has been tested and works with files generated using flowJo version 9.1 on Mac OS X. XML generated by older versions of *flowJo* on windows should work as well. We do not yet support *flowJo*'s **Chimera** XML schema, though that support will be provided in the future.

The package supports import of only a subset of the features present in a flowJo workspace. The package allows importing of sample and group names, gating hierarchy, compensation matrices, data transformation functions, a subset of gates, and population counts.

BooleanGates are now supported by flowWorkspace.

4 Data Structures

The following section walks through opening and importing a flowJo workspace.

4.1 Loading the library

Simply call:

```
> library(flowWorkspace)
```

```
Scalable Robust Estimators with High Breakdown Point (version 1.3-01)
```

The library depends on numerous other packages, including *graph, XML, Rgraphviz, flowCore, flowViz, RBGL.*

4.2 Opening a Workspace

We represent flowJo workspaces using `flowJoWorkspace` objects. We only need to know the path to, and filename of the flowJo workspace.

```
> d<-system.file("extdata",package="flowWorkspaceData");
> wsfile<-list.files(d,pattern="A2004Analysis.xml",full=T)
```

In order to open this workspace we call:

```
> ws<-openWorkspace(wsfile)
> summary(ws)
```

```
FlowJo Workspace Version  2.0
File location:  /loc/home/biocbuild/bbs-2.9-bioc/R/library/flowWorkspaceData/extdata
File name:  A2004Analysis.xml
Workspace is open.

Groups in Workspace
         Name Num.Samples
1 All Samples           2
```

We see that this a version 2.0 workspace file. It's location and filename are printed. Additionally, you are notified that the workspace file is open. This refers to the fact that the XML document is internally represented using 'C' data structures from the *XML* package. After importing the file, the workspace must be explicitly closed using `closeWorkspace()` in order to free up that memory.

4.3 Parsing the Workspace

With the workspace file open, we have not yet imported the XML document. The next step parses the XML workspace and creates R data structures to represent some of the information therein. Specifically, by calling `parseWorkspace()` the user will be presented with a list of *groups* in the workspace file and need to choose one group to import. Why only one? Because of the way flowJo handles data transformation and compensation. Each group of samples is associated with a compensation matrix and specific data transformation. These are applied to all samples in the group. When a particular group of samples is imported, the package generates a *GatingHierarchy* for each sample, describing the set of gates applied to the data (note: polygons, rectangles, quadrants, and ovals and boolean gates are supported). The set of GatingHierarchies for the group of samples is stored in a *GatingSet* object. Calling `parseWorkspace()` is quite verbose, informing the user as each gate is created. The parsing can also be done non–interactively by specifying which group to import directly in the function call (either an index or a group name). An additional optional argument `execute=T/F` specifies whether you want to load, compensate, transform the data and compute statistics immediately after parsing the XML tree.

```
> G<-parseWorkspace(ws,name=1,execute=TRUE,path=ws@path,isNcdf=FALSE,cleanup=FALSE,ke
> #Lots of output here suppressed for the vignette.
```

When isNcdf flag is set TRUE,the data is stored in ncdf format on disc.

```
> G

A GatingSet with  2  samples
1 .         FCS File:  a2004_01T2pb05i_A1_A01.fcs
        GatingHierarchy with  20  gates
2 .         FCS File:  a2004_01T2pb05i_A2_A02.fcs
        GatingHierarchy with  20  gates
```

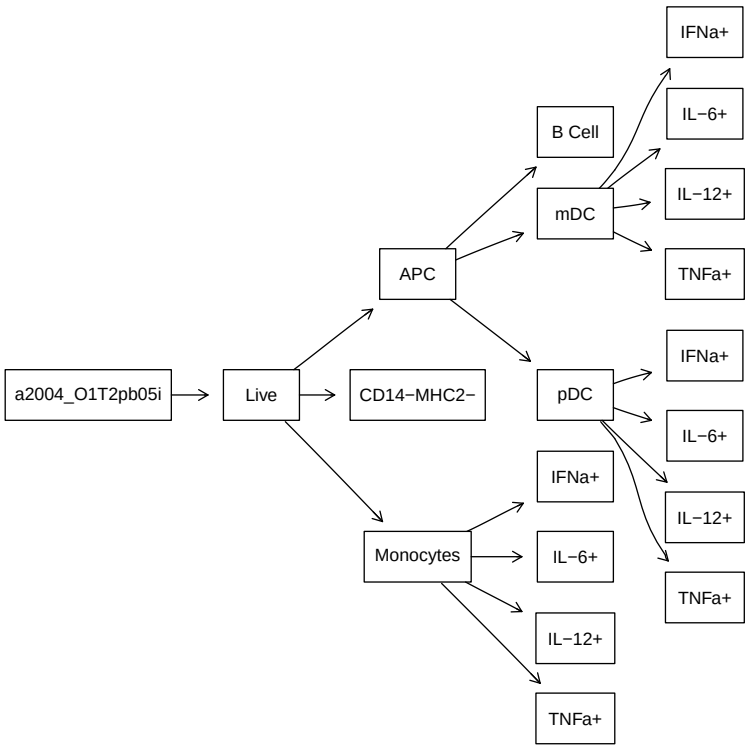
We have generated a *GatingSet* with 2 samples, each of which has 19 associated gates. Subsets of gating hierarchies can be accessed using the standard R subset syntax.

At this point we have parsed the workspace file and generate the gating hierarchy associated with each sample imported from the file. The data have been loaded, compensated, and transformed in the workspace, since we passed `execute=TRUE` to the `parseWorkspace()` function. This can also be done separately via the `execute()` method, which takes a *GatingHierarchy* and the *flowJoWorkspace* that generated it, as arguments. It returns a *GatingHierarchy* with additional data attached to each node of the hierarchy (population counts, membership indices, and a *flowFrame*.

```
> G<-lapply(G,function(x)execute(x))
```

We can plot the gating hierarchy for a given sample:

```
> plot(G[[1]])
```



We can list the nodes (populations) in the gating hierarchy:

```
> getNodes(G[[1]])

 [1] "a2004_01T2pb05i" "3.Live"          "4.APC"           "5.B Cell"
 [5] "6.mDC"           "7.IFNa+"         "8.IL-6+"         "9.IL-12+"
 [9] "10.TNFa+"        "11.pDC"          "12.IFNa+"        "13.IL-6+"
[13] "14.IL-12+"       "15.TNFa+"        "16.CD14-MHC2-"   "17.Monocytes"
[17] "18.IFNa+"        "19.IL-6+"        "20.IL-12+"       "21.TNFa+"
```

Note that the number preceding the period in the node names is just an identifier to help uniquely label populations in the gating hierarchy. It does not represent any information about population statistics. We can get a specific gate definition:

```
> getGate(G[[1]],getNodes(G[[1]])[3])

Polygonal gate '4.APC' with 14 vertices in dimensions <PerCP-CY5-5-A> and <PE-CY7-A>
```

We can extract the dimensions relating to a specific gate:

```
> getDimensions(G[[1]],getNodes(G[[1]])[3])

[1] "<PerCP-CY5-5-A>" "<PE-CY7-A>"
```

We can extract vertices of a gate:

```
> getBoundaries(G[[1]],getNodes(G[[1]])[3])

      <PerCP-CY5-5-A> <PE-CY7-A>
 [1,]        2349.993  2024.8746
 [2,]        2163.383  1575.0085
 [3,]        2240.899   992.3135
 [4,]        2349.993   793.0647
 [5,]        2585.516   696.7596
 [6,]        3315.004  1138.4273
 [7,]        3586.426  1354.9513
 [8,]        3602.373  2040.1931
 [9,]        3570.480  2256.4455
[10,]        3363.261  2318.7616
[11,]        3204.000  2240.8992
[12,]        3044.921  2209.8486
[13,]        2711.845  2070.8857
[14,]        2569.755  2055.5302
```

We can get the population proportion (relative to its parent) for a single population:

```
> getProp(G[[1]],getNodes(G[[1]])[3])

     TRUE
0.08402716
```

Or we can retrieve the population statistics for all populations in the sample:

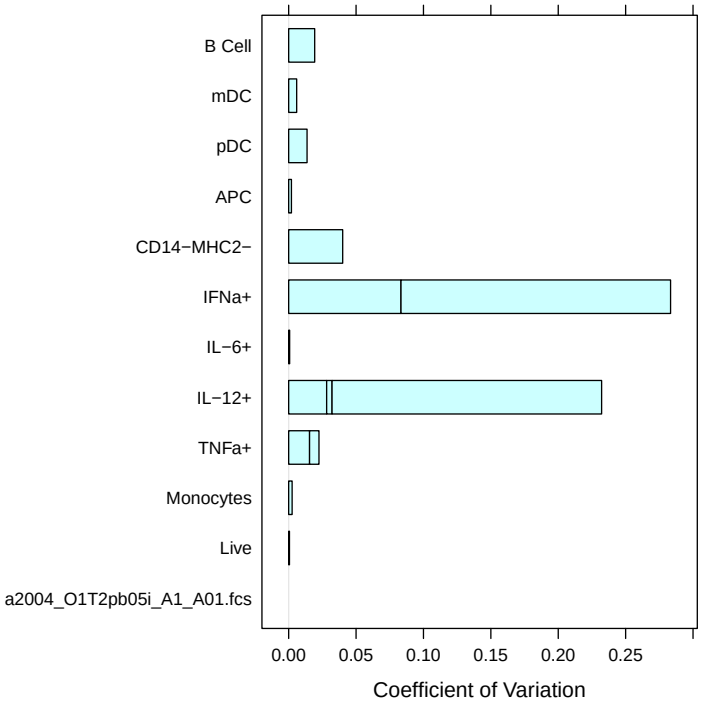
```
> getPopStats(G[[1]])
```

	flowCore.freq	flowJo.count	flowCore.count	parent.total	node
a2004_01T2pb05i_A1_A01.fcs	1.000000000	61832	61832	61832	a2004_01T2pb05i
/Live	0.800297581	49542	49484	61832	3.Live
/Live/Monocytes	0.058928138	2931	2916	49484	17.Monocytes
/Live/Monocytes/TNFa+	0.250685871	754	731	2916	21.TNFa+
/Live/Monocytes/IL-12+	0.047325103	146	138	2916	20.IL-12+
/Live/Monocytes/IL-6+	0.237654321	694	693	2916	19.IL-6+
/Live/Monocytes/IFNa+	0.003772291	13	11		
/Live/CD14-MHC2-	0.499757497	26795	24730		
/Live/APC	0.084027160	4141	4158		
/Live/APC/pDC	0.104377104	446	434		
/Live/APC/pDC/TNFa+	0.000000000	0	0		
/Live/APC/pDC/IL-12+	0.571428571	250	248		
/Live/APC/pDC/IL-6+	0.000000000	0	0		
/Live/APC/pDC/IFNa+	0.002304147	1	1		
/Live/APC/mDC	0.122174122	502	508		
/Live/APC/mDC/TNFa+	0.141732283	71	72		
/Live/APC/mDC/IL-12+	0.005905512	2	3		
/Live/APC/mDC/IL-6+	0.043307087	22	22		
/Live/APC/mDC/IFNa+	0.005905512	2	3		
/Live/APC/B Cell	0.525493025	2271	2185		

/Live/Monocytes/IFNa+	2916	18.IFNa+
/Live/CD14-MHC2-	49484	16.CD14-MHC2-
/Live/APC	49484	4.APC
/Live/APC/pDC	4158	11.pDC
/Live/APC/pDC/TNFa+	434	15.TNFa+
/Live/APC/pDC/IL-12+	434	14.IL-12+
/Live/APC/pDC/IL-6+	434	13.IL-6+
/Live/APC/pDC/IFNa+	434	12.IFNa+
/Live/APC/mDC	4158	6.mDC
/Live/APC/mDC/TNFa+	508	10.TNFa+
/Live/APC/mDC/IL-12+	508	9.IL-12+
/Live/APC/mDC/IL-6+	508	8.IL-6+
/Live/APC/mDC/IFNa+	508	7.IFNa+
/Live/APC/B Cell	4158	5.B Cell

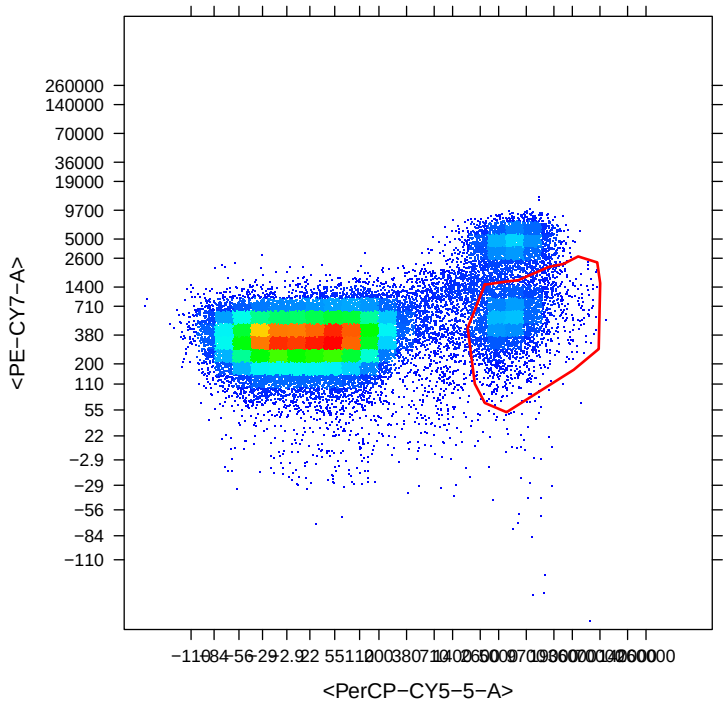
We can plot the coefficients of variation between the counts derived using flowJo and flowCore for each population:

```
> print(plotPopCV(G[[1]]))
```



We can plot individual gates: note the scale of the transformed axes.

```
> print(plotGate(G[[1]],getNodes(G[[1]])[3],lwd=2))
```



If we have metadata associated with the experiment, it can be attached to the `GatingSet`.

```
> d<-data.frame(sample=factor(c("sample 1", "sample 2")),treatment=factor(c("sample",
> G@metadata<-new("AnnotatedDataFrame",data=d)
> pData(G);
```

```
    sample treatment
1 sample 1    sample
2 sample 2   control
```

We can retrieve the subset of data associated with a node:

```
> getData(G[[1]],getNodes(G[[1]])[3]);
```

```
flowFrame object '1be493f5-51dd-4359-b2ed-524cd104eb5f'
with 4158 cells and 23 observables:
                 name  desc      range   minRange   maxRange
$P1             FSC-A  <NA> 262254.000 -111.00000 262143.000
$P2             FSC-H  <NA> 262143.000    0.00000 262143.000
$P3             FSC-W  <NA> 262143.000    0.00000 262143.000
$P4             SSC-A  <NA> 262254.000 -111.00000 262143.000
$P5             SSC-H  <NA> 262143.000    0.00000 262143.000
$P6             SSC-W  <NA> 262143.000    0.00000 262143.000
$P7       <Am Cyan-A> CD123   3661.959  435.34379   4097.303
$P8         Am Cyan-H CD123 262143.000    0.00000 262143.000
$P9  <Pacific Blue-A> IL-12   3927.974  169.60860   4097.582
$P10   Pacific Blue-H IL-12 262143.000    0.00000 262143.000
$P11          <APC-A> CD11c   4405.818 -308.01302   4097.805
$P12            APC-H CD11c 262143.000    0.00000 262143.000
$P13      <APC-CY7-A>  IL-6   3714.446  382.93207   4097.378
$P14        APC-CY7-H  IL-6 262143.000    0.00000 262143.000
$P15    <Alexa 700-A>  TNFa   3712.753  384.62271   4097.376
$P16      Alexa 700-H  TNFa 262143.000    0.00000 262143.000
$P17         <FITC-A>  IFNa   4180.519  -82.81306   4097.706
$P18           FITC-H  IFNa 262143.000    0.00000 262143.000
$P19  <PerCP-CY5-5-A> MHCII   4942.398 -844.59317   4097.805
$P20    PerCP-CY5-5-H MHCII 262143.000    0.00000 262143.000
$P21       <PE-CY7-A>  CD14   4942.398 -844.59317   4097.805
$P22         PE-CY7-H  CD14 262143.000    0.00000 262143.000
$P23             Time  <NA>   9918.400   89.00000  10007.400
322 keywords are stored in the 'description' slot
```

Or we can retrieve the indices specifying if an event is included inside or outside a gate using:

```
> getIndices(G[[1]],getNodes(G[[1]])[3])
```

The indices returned are relative to the parent population (member of parent AND member of current gate), so they reflect the true hierarchical gating structure.

If we wish to do compensation or transformation manually, we can retrieve all the compensation matrices from the workspace:

```
> C<-getCompensationMatrices(ws);
> C
```

```
$`A2004-A2005_06i`
               Am Cyan-A Pacific Blue-A    APC-A APC-CY7-A Alexa 700-A
Am Cyan-A        1.00000        0.04800 0.000000    0.0000     0.00000
Pacific Blue-A   0.38600        1.00000 0.000529    0.0000     0.00000
APC-A            0.00642        0.00235 1.000000    0.0611     0.19800
APC-CY7-A        0.03270        0.02460 0.084000    1.0000     0.02870
Alexa 700-A      0.07030        0.05800 0.016200    0.3990     1.00000
FITC-A           0.74500        0.02090 0.001870    0.0000     0.00000
PerCP-CY5-5-A    0.00368        0.00178 0.015300    0.0269     0.07690
PE-CY7-A         0.01330        0.00948 0.000951    0.1380     0.00182
                  FITC-A PerCP-CY5-5-A PE-CY7-A
Am Cyan-A       0.028500       0.00104  0.00000
Pacific Blue-A  0.000546       0.00000  0.00000
APC-A          -0.000611       0.00776  0.00076
APC-CY7-A       0.002690       0.00304  0.01010
Alexa 700-A     0.001530       0.10800  0.00679
FITC-A          1.000000       0.04180  0.00281
PerCP-CY5-5-A   0.000000       1.00000  0.07030
PE-CY7-A        0.002340       0.03360  1.00000
```

Or we can retrieve transformations:

```
> T<-getTransformations(ws)
> names(T)

[1] "InputParameterTransform_Gain1_Offset1"
[2] "A2004-A2005_06i"
[3] "InputParameterTransform_Gain1_Offset1262144"

> names(T[[1]])

[1] "InputParameterTransform_Gain1_Offset1"
[2] "InputParameterTransform_Gain1_Offset1262144"

> T[[1]][[1]]

function (x, deriv = 0)
{
    deriv <- as.integer(deriv)
    if (deriv < 0 || deriv > 3)
        stop("'deriv' must be between 0 and 3")
```

```
    if (deriv > 0) {
        z0 <- double(z$n)
        z[c("y", "b", "c")] <- switch(deriv, list(y = z$b, b = 2 *
            z$c, c = 3 * z$d), list(y = 2 * z$c, b = 6 * z$d,
            c = z0), list(y = 6 * z$d, b = z0, c = z0))
        z[["d"]] <- z0
    }
    res <- .C(C_spline_eval, z$method, as.integer(length(x)),
        x = as.double(x), y = double(length(x)), z$n, z$x, z$y,
        z$b, z$c, z$d, PACKAGE = "stats")$y
    if (deriv > 0 && z$method == 2 && any(ind <- x <= z$x[1L]))
        res[ind] <- ifelse(deriv == 1, z$y[1L], 0)
    res
}
<bytecode: 0x5b111a8>
<environment: 0x6d72278>
```

`getTransformations` returns a list, each element of which corresponds to a transformation applied to a group of samples. The transformation is presented as a list of functions to be applied to different dimensions of the data. Above, the transformation is applied to all samples of the group and for each sample in the group, the appropriate dimension is transformed using a channel–specific function from the list.

The list of samples in a workspace can be accessed by:

```
> getSamples(ws);

  sampleID            name count compID pop.counts
1        1 a2004_01T2pb05i 61832      1         19
2        2 a2004_01T2pb05i 45363      1         19
```

And the groups can be accessed by:

```
> getSampleGroups(ws)

    groupName groupID sampleID
1 All Samples       0        1
2 All Samples       0        2
```

The `compID` column tells you which compensation matrix to apply to a group of files, and similarly, based on the name of the compensation matrix, which transformations to apply.

4.4 Converting to flowCore Objects

You may want to convert the imported workspace into `flowCore` objects, such as workflows. We provide this functionality via the `flowWorkspace2flowCore` function.

`flowWorkspace2flowCore` extracts the compensation matrices,transformation functions and all the gates from GatingHierarchies generated by flowWorkspace package and converts them to the respective views and actionItems of workFlows defined by flowCore package. It takes a gatingHierarchy, flowJoWorkspace or GatingSet as the input, and returns one or multiple workflows as the result, depending on whether the gating hierarchies for each sample (including gate coordinates) are identical.

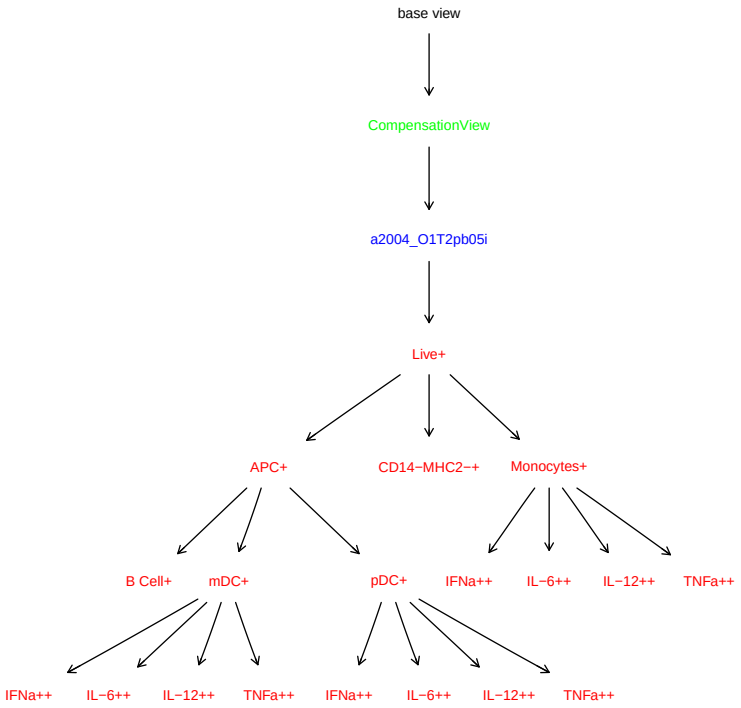
```
> wfs<-flowWorkspace2flowCore(G,path=ws@path);
> wfs

[[1]]
A flow cytometry workflow called 'default'
The following data views are provided:

 Basic view 'base view'
 on a flowSet
 not associated to a particular action item

        View 'CompensationView'
         on a flowSet linked to
         compensation action item 'action_defaultCompensation'

            View 'a2004_01T2pb05i'
             on a flowSet linked to
             transform action item 'action_defaultTransformation'

                View '3.Live+'
                 on a flowSet linked to
                 gate action item 'action_3.Live'

                    View '4.APC+'
                     on a flowSet linked to
                     gate action item 'action_4.APC'

                        View '5.B Cell+'
```

```
 on a flowSet linked to
 gate action item 'action_5.B Cell'

View '6.mDC+'
 on a flowSet linked to
 gate action item 'action_6.mDC'

     View '7.IFNa++'
      on a flowSet linked to
      gate action item 'action_7.IFNa+'

     View '8.IL-6++'
      on a flowSet linked to
      gate action item 'action_8.IL-6+'

     View '9.IL-12++'
      on a flowSet linked to
      gate action item 'action_9.IL-12+'

     View '10.TNFa++'
      on a flowSet linked to
      gate action item 'action_10.TNFa+'

View '11.pDC+'
 on a flowSet linked to
 gate action item 'action_11.pDC'

     View '12.IFNa++'
      on a flowSet linked to
      gate action item 'action_12.IFNa+'

     View '13.IL-6++'
      on a flowSet linked to
      gate action item 'action_13.IL-6+'

     View '14.IL-12++'
      on a flowSet linked to
      gate action item 'action_14.IL-12+'

     View '15.TNFa++'
```

```
                                    on a flowSet linked to
                                    gate action item 'action_15.TNFa+'

                                View '16.CD14-MHC2-+'
                                 on a flowSet linked to
                                 gate action item 'action_16.CD14-MHC2-'

                                View '17.Monocytes+'
                                 on a flowSet linked to
                                 gate action item 'action_17.Monocytes'

                                     View '18.IFNa++'
                                      on a flowSet linked to
                                      gate action item 'action_18.IFNa+'

                                     View '19.IL-6++'
                                      on a flowSet linked to
                                      gate action item 'action_19.IL-6+'

                                     View '20.IL-12++'
                                      on a flowSet linked to
                                      gate action item 'action_20.IL-12+'

                                     View '21.TNFa++'
                                      on a flowSet linked to
                                      gate action item 'action_21.TNFa+'
>
```

`plotWf` plots the workflow tree

```
> plotWf(wfs[[1]])
```



Finally, when we are finished with the workspace, we close it:

```
> closeWorkspace(ws);
> ws

FlowJo Workspace Version  2.0
File location:  /loc/home/biocbuild/bbs-2.9-bioc/R/library/flowWorkspaceData/extdata
File name:  A2004Analysis.xml
Workspace is closed.
```

4.5 Exporting to FlowJo OSX 9.2

The `exportAsFlowJoXML` function can be used to export a `flowCore::workFlow` as an XML workspace for FlowJo 9.2 OSX. If flowWorkspace has been used to import an existing FlowJo workspace, `flowWorkspace2flowCore` can be used to obtain a `workFlow` for exporting. Currently this function can export one workFlow at a time.

4.6 Additional Important Notes

4.6.1 NetCDF Support

If you have particularly large data files (millions of events), then you will want to make use of the netCDF framework. To do so, you will have to install the netcdf4 C library (available at `http://www.unidata.ucar.edu/downloads/netcdf/index.jsp`), and build it with HDF5 support.You will also need the R library ncdf4. To use NetCDF, pass isNcdf=TRUE to parseWorkspace. flowWorkspace will create one netcdf file for the processed data, and additional netcdf files (one per sample) containing the event memberships for each gate in each sample. To build flowWorkspace to use netcdf, you may need to run autoconf in the top–level directory of the untarred flowWorkspace source directory, before installing with R CMD INSTALL.

4.6.2 Known Bugs

Importing flowJo transformations. We have made every effort to support the importing of flowJo's data transformations. Sometimes, however, flowWorkspace may have difficulty identifying the correct transofrmation to apply to your data. There are several things you can do:

- Visualize your data after import using plotPopCV(). This will tell you if there is a discrepancy between the flowJo counts and the flowCore counts for individual populations. Keep in mind that tare populations can differ by a few cells and still show large coefficients of variation.
- If you are having difficulty importing your data due to a transformation problem, ensure that you are using either a log transform, or flowJo's "custom biexponential transform". The latter provides an explicit mapping between transformations, compensation matrices, and flow parameters. Outside of these two cases, success or failure appears to be dependent on the version of flowJo that was used to create the original workspace.

5 Troubleshooting

If this package is throwing errors when parsing your workspace, and you are certain your workspace is version 2.0, contact the package author. If you can send your workspace by email, we can test, debug, and fix the package so that it works for you. Our goal is to provide a tool that works, and that people find useful.

6 Future Improvements

We are working on support for flowJo XML workspaces exported from the Windows version of flowJo. Efforts are underway to integrate GatingSet and GatingHierarchy objects more closely with the rest of the flow infrastructure.