Programming with Maltcms 0.95

Nils Hoffmann
Nils.Hoffmann@CeBiTec.Uni-Bielefeld.DE

12. November 2009

Inhaltsverzeichnis

1 Introduction

The following Tutorial was written as an introduction to the Java based framework Maltcms (Modular Application Toolkit for Chromatography Mass Spectrometry). It tries to give an overview of key classes and concepts, as well as in depth information, where necessary.

1.1 Audience

The target audience of this tutorial are typically programmers or scientists, wishing to extend the functionality of Maltcms, or wishing to incorporate Maltcms functionality into their own projects.

2 Components

The following section gives a short overview of the core components, mainly situated in the generic package `cross.*` (Common Runtime Object Support System). These components provide basic datastructures and infrastructure for logging, IO and other things.

2.1 A document tree datastructure: IFileFragment and FileFragment

[1] Maltcms makes use of the netcdf[2] file format which is based on an abstract model for scientific data. This model allows for naming of variables, attributes and dimensions within one file, thus creating self-describing data. There is also a possibility for using structures and groups within such files. However, in Maltcms only a subset of this model is used, and mirrored as an extended, internal data structure, which allows for serialization to XML and lazy (on demand) loading of array data, as well as indexed reading in row compressed storage (rcs) format of one array with indices stored in another array.[3] The internal structure can be imagined as a document tree, where the root of the tree is an instance of `IFileFragment`. It is possible to add instances of `IVariableFragment` to the root, as well as removing them. Each `IVariableFragment` has different attributes, which can be altered.

Arbitrary input file formats are mapped to the internal array style format by corresponding `cross.io.IDataSource` implementations. Such a data file must provide methods for reading single array data, as well as for reading scan oriented indexed arrays. The implementation is required to react on passed in references to objects of type `cross.datastructures.fragments.IVariableFragment`, which provide a name for the data part associated with that name. The names of variables mimic those defined in the AIA/ANDI-MS standard, so that once the internal data structure has been established from a different input file via an appropriate `cross.io.IDataSource` implementation, the data can be saved to an AIA/ANDI-MS compatible netcdf file.

2.1.1 Referencing previous IFileFragment Objects

2.2 Manipulating document trees: AFragmentCommand

The classes extending `cross.commands.fragments.AFragmentCommand` build the core classes of a processing pipeline. Each such class must provide a method

[1] found in package `cross.datastructures.fragments`

[2] `http://www.unidata.ucar.edu/software/netcdf/`

[3] The associated classes can be found in the package `maltcms.datastructures.fragments`. The package name will be omitted from now on, unless necessary for unique identification of classes.

public TupleND<IFileFragment> apply(TupleND<IFileFragment> t) to receive a tuple of `IFileFragment` objects and return them possibly processed or replaced.

As an example of how such a `AFragmentCommand` is created, the class `Normalizer` will be used as an example. Let's focus on the necessary imports first:

```
/* The package declaration is placed at the top of the File
 * Since this example will be application domain specific, we
 * place it under maltcms.commands.fragments
 */
package maltcms.commands.fragments;
// The most basic imports are
import cross.datastructures.fragments.IFileFragment;
import cross.datastructures.fragments.FileFragment;
import cross.datastructures.tuple.TupleND;
// In order to use the logging and configuration infrastructure of Maltcms,
// we also need the following imports
import cross.Logging;
import org.slf4j.Logger;
import org.apache.commons.configuration.Configuration;
// We also add an import to throw Exceptions from not yet implemented methods
import maltcms.exception.NotImplementedException;
```

After having imported the basic dependencies necessary for our Normalizer, we can begin describing our class.

```
// Now we can start describing Normalizer, which extends
// FragmentCommand
public class Normalizer extends AFragmentCommand {
// Define some private variables
    private boolean normalize_mass_channels = false;
    private boolean normalize_scan_intensities = false;
    private Logger log = Logging.getLogger(this.getClass());
// Implement the apply method of FragmentCommand
    public TupleND<IFileFragment> apply(TupleND<IFileFragment> t) {
// Let it throw a NotImplementedException for now:
        throw new NotImplementedException();
    }
    ...
```

```
public class Normalizer extends AFragmentCommand {
   ...
// Next, we should override the configure method of
// Configurable to allow the Factory to configure objects of this class
// at runtime. Since Java version 1.5, overridden methods should be annotated
// with @Override.
    @Override\\
    public void configure(Configuration cfg) {
// We will set the private variables in here and publish log messages in
// info-mode, the second parameter in cfg.getBoolean(String s, boolean b)
// is the default value, if s is not found within the configuration.
        this.normalize_mass_channels=cfg.getBoolean("normalize.
            mass_channels",true);
// The braces  within the logging statement are used for pattern
// substitution, to prevent costly variable expansion/evaluation if we are in a
// different logging mode.
        log.info("Normalizing mass channels = {}",this.
            normalize_mass_channels);
        this.normalize_scan_intensities=cfg.getBoolean("normalize.
            scan_intensities",true);
        log.info("Normalizing scan intensities = {}",this.
            normalize_scan_intensities);
```

```
19      }
20  }
```

In order to put some functionality into our class, we will begin implementing the `apply` method.

```
1  public TupleND<IFileFragment> apply(TupleND<IFileFragment> t)
       {
2  // Iterate over all IFileFragment objects in t
3       for(IFileFragment ff:t){
4  // Create a IFileFragment with default name to take the
5  // results of operation
6           IFileFragment work = FragmentTools.create(this.
                getClass());
7  // Query for a specific variable
8           if(ff.hasChildren("intensity_values","scan_index")){
9  // intensity_values is an indexed variable
10              VariableFragment intensity_values = ff.getChild("
                    intensity_values");
11              VariableFragment scan_index = ff.getChild("
                    scan_index");
12              intensity_values.setIndex(scan_index);
13          }
14      }
15  }
```

Now that we have some VariableFragments ready, we can try to read their associated arrays.

```
1  // Returns the scans contained in intensity_values.
2  // Read access uses the index variable scan_index
3  ArrayList<Array> intens_scan_arrays = intensity_values.
       getIndexedArray();
4  // Traversing the scans
5  for(Array arr:intens_scan_arrays) {
6  //we can try to cast an array to a more concrete type
7       if(arr instanceof ArrayInt.D1){
8           ArrayInt.D1 scan_i = (ArrayInt.D1)arr;
9  //read value 100 of scan
10          int value =  scan_i.get(100);
11      }
12 // otherwise, we can always use an IndexIterator to traverse all values
13      IndexIterator iter = arr.getIndexIterator();
14      while(iter.hasNext()) {
15          double d = iter.getDoubleNext();
16      }
17 }
18 // We can also read the array of intensity_values as a whole,
19 // in it's row compressed storage representation
20 Array intens_values_array = intensity_values.getArray();
21 // We can of course read scan_index as well
22 Array a = scan_index.getArray();
23 // If the type of an array and it's shape are known in advance,
24 // we can directly cast to the appropriate type.
25 // This allows for direct element access, without using an Index.
```

```
26 ArrayInt.D1 scan_index_array = ((ArrayInt.D1)a);
```

2.3 Building a pipeline with CommandSequence

Command sequence is a collection type class, allowing a special kind of iteration. Initialization is performed by passing in a List of `FragmentCommand` objects and input `IFileFragment` objects. Then, upon each invocation of the usual

`if(hasNext)->next()`

paradigm on iterable objects, the output of the last active `FragmentComand` is pushed as input to the next `FragmentCommand`. It is thus possible to build processing chains[4], which start for example with loading of default values, then preprocessing, then continue applying a feature detection algorithm and end with a visualization. The file `cfg/default_pipeline.properties` contains a key to set the elements of the pipeline used in left to right order called `pipeline`.

The simple pipeline scheme has some limitations though, if for example a `FragmentCommand` creates additional data, which is needed by `FragmentCommand` later in the pipeline. For this case, we currently use the configuration, obtainable via `Factory.getConfiguration()`, adding a new key-value mapping pointing e.g. to the file representation of the data.

```
1 //The following line would load FragmentCommand objects
2 //as given under property anchors.class, loading possibly
    available retention
3 //indices, then would run the default.varloader, which loads
    all variables listed under
4 //default.variables. Then, the dense arrays are produced from
    row compressed storage arrays
5 //and finally, pairwise distances between the arrays are
    calculated
6 pipeline = ${anchors.class},${default.varloader},${dense.
    arrays},${pairwise.distances}
```

2.4 Manipulating Arrays: ArrayCommand

3 Tools

3.1 ArrayTools

3.2 MaltcmsTools

3.3 FileTools

4 CLI, Configuration and Logging Infrastructure

Maltcms uses pre-existing software solutions to provide a common logging, command line and configuration system. Those systems are in wide use and thus rather bug free.

[4] Even for batch processing, since n-Tuples of `IFileFragment` serve as input.

4.1 Command Line Interface with Jakarta Commons CLI

The class `apps.Maltcms` uses the command line interface provided by Apache Jakarta Commons CLI and defines custom command line parameters, which can be inspected by typing the usual `-?,-help,-h` options after the program name.

4.2 Configuration with Jakarta Commons

Configuration in Maltcms is usually done in java properties file format, e.g. `PROPERTYNAME = VALUE`. The main configuration options are stored in `cfg/default.properties`. Configuration files can be aggregated by including other configuration files: `include = other.properties`.

4.3 Logging with Simple Logging Facade for Java: SLF4J

The simple logging facade allows easy exchange of the underlying logging system. Currently, apache log4j is used, but potentially any other compatible logging framework, such as the jdk's own logging could be used.

5 Supporting Scripts

5.1 `scripts/maltcms.sh`

```
#!/bin/bash
export JARCH=""
export JBIN="\/vol\/java-1.6.0\/bin"
export EXEC="apps.Maltcms"
export MXSIZE="2G"
export MSSIZE="256M"
export MALTCMSARGS=""
export PROFILE=""
export USRCLSPATH=""

if [ -z "$MALTCMSDIR" ]; then
    echo "Please enter path to Maltcms installation or add
        MALTCMSDIR to your bash profile:"
    read MALTCMSUSRDIR;
    if [ -z "$MALTCMSUSRDIR" ]; then
        echo "No user defined directory for Maltcms
            installation entered, no default g____iven,
            exiting!";
    exit 1;
fi
    else
        MALTCMSUSRDIR=$MALTCMSDIR;
fi

LOG4J_LOCATION="-Dlog4j.configuration=file://$MALTCMSUSRDIR/
    cfg/log4j.properties"

#Check if javahome exists => contains path to java
if [ -f $MALTCMSUSRDIR/javahome ]; then
```

```
26      echo "File javahome exists";
27  else
28      $MALTCMSUSRDIR/scripts/setJavaHome.sh;
29  fi
30
31  export JAVA_HOME="$(cat $MALTCMSUSRDIR/javahome)";
32  echo -e "Using the following java location: $JAVA_HOME\n"
33
34  function printHelp {
35              echo -e "Usage: $0 [-64] [-mx ARG] [-ms ARG] [-
                    exec ARG|--execute ARG] [-?|--help] [--
                    MALTCMSARGS]"
36              #echo -e "Usage: $0 [-cp] [-64] [-mx ARG] [-exec
                    ARG|--execute ARG] [-?|--help] [-- MALTCMSARGS
                    ]"
37              #echo -e "\t-cp -> builds classpath from jars in
                    lib/"
38              echo -e "\t-64 -> uses 64bit jvm"
39              echo -e "\t-mx ARG -> uses -XmxARG as maximum
                    heapsize"
40              echo -e "\t-ms ARG -> uses -XmsARG as minimum
                    heapsize"
41              echo -e "\t-exec ARG|--execute ARG -> execute the
                    given base class,"
42                  echo -e "\t\t\tif it contains a main method"
43              echo -e "\t\t\te.g. '-exec mypackage/MyClass'"
44              echo -e "\t-?|--help -> display this help"
45              echo -e "\t-- MALTCMSARGS -> hands all arguments
                    following '--' over to Maltcms"
46              exit -1
47  }
48
49  if [ $# -eq 0 ]; then
50      printHelp $1
51  fi
52
53  while [ $# -gt 0 ]; do
54      case "$1" in
55          #-cp)    do this no matter what happens, ensures a
                clean CP
56          #    scripts/buildCP.sh
57          #    #echo $CLASSPATH
58          #    ;;
59          -64)
60              export JARCH="-d64"
61              ;;
62          -exec|--execute)
63              shift
64              export EXEC="$1"
65              ;;
66          -profile)
67              export PROFILE="-agentlib:hprof=heap=sites,file=
                    hprof.out,format=b,cpu=samples,interval=20"
68              ;;
```

```
69          -mx)
70              shift
71              export MXSIZE="$1"
72              ;;
73          -ms)
74              shift
75              export MSSIZE="$1"
76              ;;
77          --)
78              echo "Running a $JARCH VM with -Xmx $MXSIZE"
79              shift
80              #Check for clspath file
81              if [ -f $MALTCMSUSRDIR/clspath ]; then
82                  echo "File clspath exists";
83              else
84                  $MALTCMSUSRDIR/scripts/buildCP.sh
85              fi
86              USRCLSPATH="$CLASSPATH:$(cat $MALTCMSUSRDIR/
                    clspath)"
87              echo -e "Passing args to $EXEC"
88              echo -e "$@"
89              sleep 1
90              $JAVA_HOME/bin/java -cp $USRCLSPATH $PROFILE -
                    Xms$MSSIZE -Xmx$MXSIZE $JARCH $LOG4J_LOCATION
                    $EXEC "$@"
91              exit $?
92              ;;
93          -"?"|--help)
94              printHelp $0
95              ;;
96
97      esac
98      shift
99  done
100
101 exit 1
```

5.2 scripts/buildCP.sh

```
1  #!/bin/bash
2  if [ "$1" != "" ]; then
3      FPATH=$1
4  else
5      FPATH=$(pwd)/scripts
6  fi
7  if [ -e clspath ]; then
8      rm clspath
9  fi
10 for i in `ls $(echo -e $FPATH | sed -e s/scripts/lib/g)/*.jar`
11 do
12     echo -n ":$i" >> clspath;
13 done
```

```
#for i in `ls $(echo -e $FPATH | sed -e s/scripts/cfg/g)/*.
    properties`
#do
#    echo -n ":$i" >> clspath;
#done
#echo -e $FPATH
PATH=`echo -e $FPATH | sed -e s/scripts/bin/g`
echo -e ":$PATH" >> clspath
#export CLASSPATH="$CLASSPATH$(cat clspath)"
#echo $CLASSPATH
```

5.3 scripts/buildAntCP.sh

```
#!/bin/bash
if [ "$1" != "" ]; then
    FPATH=$1
else
    FPATH=$(pwd)/scripts
fi
if [ -e antclspath ]; then
    rm antclspath
fi
for i in `ls $(echo -e $FPATH | sed -e s/scripts/lib/g)/*.jar`
do
    echo -en "<pathelement location=\"$i\"/>\n" >> antclspath;
done
```

5.4 scripts/setJavaHome.sh

```
#!/bin/bash
ARCH=`uname -a`
OSTYPE=`uname -s`
#echo -e "Running on $ARCH\n"
JAVA_HOME=""
case "$OSTYPE" in
    SunOS)
        JAVA_HOME="/vol/java-1.6/"
        ;;
    Darwin)
        JAVA_HOME="/System/Library/Frameworks/JavaVM.framework
            /Versions/1.6.0/Home/"
        ;;
    Linux)
        JAVA_HOME="/usr/lib/jvm/java-6-sun/"
        ;;
    *)
        echo "Unknown OS $OSTYPE, please set the path to your
            Java-Installation by hand."
        echo "Save it in a file called javahome within maltcms
            basedir"
        ;;
```

```
esac
echo -e "$JAVA_HOME" > javahome
echo $JAVA_HOME
```

5.5 scripts/updateBuildNumber.sh

```
#!/bin/bash
REVISION=`svn info | grep -e Revision | sed -e "s/Revision: //"`
DATE=`date +%Y_%b_%d`
sed -i -e "s/application.version.revision=-r[0-9]*/application.version.revision=-r$REVISION/" cfg/application.properties
sed -i -e "s/application.build.date=-[0-9A-Za-z_]*/application.build.date=-$DATE/" cfg/application.properties
cut -f 2 -d= cfg/application.properties > versionfile;
grep -v "\\$" versionfile > tmpversionfile;
paste -s -d\\0 tmpversionfile > versionfile;
rm tmpversionfile;
sed -e "s/PROJECT_NUMBER\W*=.*/PROJECT_NUMBER=$(cat versionfile)/g" -i Doxyfile
cat versionfile;
```