The Modular Application Toolkit for Chromatography-Mass Spectrometry

User Guide

Version 1.2.1-SNAPSHOT

Nils Hoffmann

May 17, 2013

Preface

Intended Audience

This guide is aimed at novice as well as at experienced users of the Modular Application Toolkit for Chromatography-Mass Spectrometry (Maltcms) wishing to get a step-by-step introduction to setting up and running processing workflows for raw data from chromatography-mass spectrometry experiments.

Additionally, this guide provides programmers and power-users with the required knowledge for extending Maltcms with custom functionality.

License agreement

Availability

The Maltcms distribution is available from maltcms.sf.net.
A selection of raw sample data, required for the instructional examples, is available here.

Acknowledgements

We would like to thank the developers and testers of Maltcms, namely Sören Müller, Rolf Hilker, Mathias Wilhelm and Kai-Bernd Stadermann for their contributions.

We would further like to thank Professor Dr. Jens Stoye of the Genome Informatics Group and Professor Dr. Karsten Niehaus of the Metabolome and Proteome Group at Bielefeld University for their expert advice and continued support.

```java
        System.out.println("Workflow using commands " + dw.getCommandSequence().getCommands());
        System.out.println("Workflow using inputFiles " + dw.getCommandSequence().getInput());
        System.out.println("Workflow using outputDirectory " + dw.getOutputDirectory());
        return dw;
    }

    /**
     *
     * @param args
     */
    public static void main(String[] args) {
        //Download the test files from
        //http://sf.net/projects/maltcms/files/maltcms/example-data/
        List<File> inputFiles = Arrays.asList(
                new File("glucoseA.cdf"),
                new File("glucoseB.cdf"),
                new File("mannitolA.cdf"),
                new File("mannitolB.cdf"));
        List<IFragmentCommand> cmds = Arrays.asList(new IFragmentCommand[]{
                    new DefaultVarLoader(),
                    new DenseArrayProducer(),
                    new TICPeakFinder(),
                    new PeakCliqueAlignment()});
        IWorkflow w = createWorkflow(
```

Listing 1: Programmatic assembly of a workflow

Contents

1. Introduction

1.1. Getting Started

Part I.

Using Maltcms

2. Installation

Maltcms provides a command line interface (CLI) for direct invocation from a system's command prompt. Maltcms is usually called by a platform dependent wrapper script that is located below Maltcms' installation location:

- Microsoft Windows: bin/maltcms.bat
- Mac OS X: bin/maltcms.command
- Linux/Unix: bin/maltcms.sh

These wrapper scripts provide additional options and comfort functions over the default Maltcms command line and are thus recommended for general use.

2.1. Maltcms Command Line Arguments

The following basic command line arguments apply to all operating system platforms:

```
-?                                       display this help
-a <file1,...>                           alignment anchor files (without path prefix, assumes ↓
   ↪ location as given with option -i)
-b,--createBeanXml <class1,class2, ...>  Creates a fragmentCommands.xml file in spring format ↓
   ↪ for all instances of cross.commands.fragments.AFragmentCommand
-c <file>                                configuration file location
-e <file:///path1/,...>                  extension locations (directories or .jar files)
-f <file1,...>                           input files (wildcard, e.g. *.cdf, file name only if ↓
   ↪ -i is given, or full path)
-h                                       display this help
-i <dir>                                 input base directory
-l <class1,...>                          print available service providers (e.g. ↓
   ↪ cross.commands.fragments.AFragmentCommand)
-o <dir>                                 target directory for all output
-p                                       print resolved configuration
-r                                       recurse into input base directory
-s,--showProperties <class1,class2, ...> Prints the properties available for configuration of ↓
   ↪ given classes
```

2.1.1. Handling Files

-r
-i
-f

2.1.2. Handling Output

Output Base-Directory Location

-o

Changing the Output Directory Hierarchy

-DomitUserTimePrefix
-Doutput.overwrite=true

Postprocessing of Workflow Results selecting and zipping processing results

2.2. Platform Dependent Wrapper Scripts

2.2.1. MS Windows

2.2.2. MacOS X

2.2.3. Linux/Windows

3. Usage Scenarios

3.1. One-Dimensional Chromatography

3.1.1. Pyrolysis–Gas Chromatography

3.1.2. Gas Chromatography–Mass Spectrometry

3.1.3. Liquid Chromatography–Mass Spectrometry

3.2. Two-Dimensional Chromatography

3.2.1. Modulation Methods

3.2.2. GCxGC–Mass Spectrometry

3.2.3. LCxLC–Mass Spectrometry

3.3. Putative Compound Identification

3.4. Quantification

Part II.

Developing Maltcms

4. Concepts

4.1. Testing Framework

Maltcms' testing infrastructure is based on JUnit 4.10. The maltcms-test-tools folder below the main Maltcms Maven project contains two essential sub-projects: maltcms-test-defaults and maltcms-test-data that are built and distributed individually from Maltcms. Both of these dependencies should be included on projects whishing to use the Maltcms testing infrastructure. The parent pom of Maltcms does this automatically for all its child projects.

4.1.1. Plain Unit Testing

Simple unit tests in Maltcms are easy to define, following the JUnit cookbook. First, create a new class below the src/test/java folder of your Maven project. Then, for each behaviour of your class that you want to test, create one declaratively named method, optionally with a test prefix to better distinguish it from auxiliary code. Add the @org.junit.Test annotation before or in the line above the method's signature. Within the method, implement your testing logic, using the usual Assert.assert* methods. An example is given in listing 4.1, assuming the class MyFirstClass is located in src/main/java/my/first/module. Run mvn test on your project and watch for the test summary output.

Listing 4.1: Example of a plain unit test

```
package my.first.module;

import org.junit.Test;
//static import since Java 1.5
import static junit.framework.Assert.*;

public class MyUnitTest {
        @Test
        public void testMyAssertion() {
                MyFirstClass mfc = new MyFirstClass();
                assertTrue(mfc.isReady());
        }
}
```

4.1.2. Integration Testing

The maven-failsafe-plugin provides the execution logic for integration tests. If you want to define your unit test to be an integration test, simply annotate your class with @Category(IntegrationTest.class), where Category is in the package org.junit.experimental.categories and IntegrationTest is in the maltcms.test package.

Integration tests are skipped by default, as they may be long-running and resource-consuming. To enable them, add -DskipITs=false to your Maven command line, e.g. mvn -DskipITs=false install. Details on the specific configuration for Maltcms can be found in Maltcms' pom.xml.

4.1.3. Using the Maltcms Testing Infrastructure

The module maltcms-test-defaults contains a number of convenience classes to make testing of Maltcms commands and application a lot easier. Within the package maltcms.test, there are currently three children. IntegrationTest is the marker interface used to identify tests that should be run by the maven-failsafe-plugin mentioned in section 4.1.2. SetupLogging is a TestWatcher that can be included in test classes in order to set up log4j compatible logging in conjunction with the slf4j binding for log4j. To do so, simply create a new member variable:

```
@Rule
private final SetupLogging sl = new SetupLogging();
```

Listing 4.2: Abstract base test class showing logging configuration and annotation.

```
import org.apache.log4j.Level;
import org.junit.Rule;
import org.junit.rules.TemporaryFolder;

/**
 *
 * @author Nils.Hoffmann@cebitec.uni-bielefeld.de
 */
@Slf4j
```

If you also use lombok, you can annotate your test case classes with the @Slf4j annotation and have a readily configured logging system available for your unit tests. To see this in action, have a look at the abstract test base classAFragmentCommandTest. A short excerpt is given here for reference:

Appendix

Listings