Manipulate Tabular Data

Introduction

This guide describes how to create a simple Java Algorithm which can read and write into a tabular data format. You also will learn how to mutate the parameters. Please refer to Supported Data Formats for other available data formats.

We will read in the nameWithFavorNumber.csv file which contains tabular data. During this tutorial you will retrieve data from the columns of the input file (name and favor number) and print that data in the CIShell console. In the **Generate a Tabular Output for the Algorithm** section you will create a sample output table.

Prerequisites

You should understand the CIShell Basics. Make sure that Sci2 is installed and you have set your target platform in Eclipse as described in Setting Up the Development Environment. If you are not familiar with Sci2 development you should probably read Developing a Single Plugin and complete the Hello World tutorial.

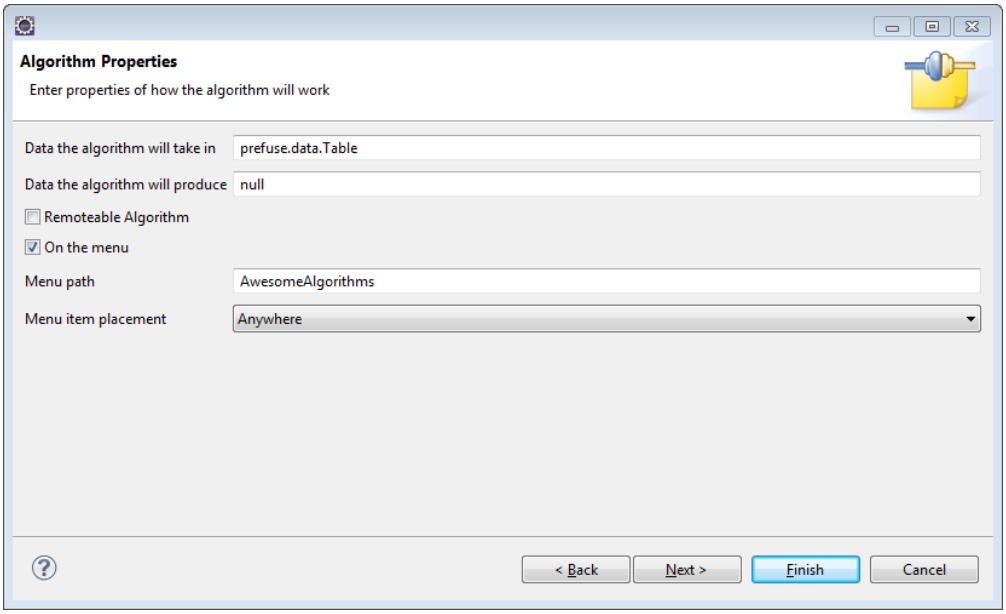
Section Table of Contents

- Creating an Algorithm with Tabular Data as Input
- Mutating and Preserving Parameters
- Generate a Tabular Output for the Algorithm
- Resolving Missing Dependencies

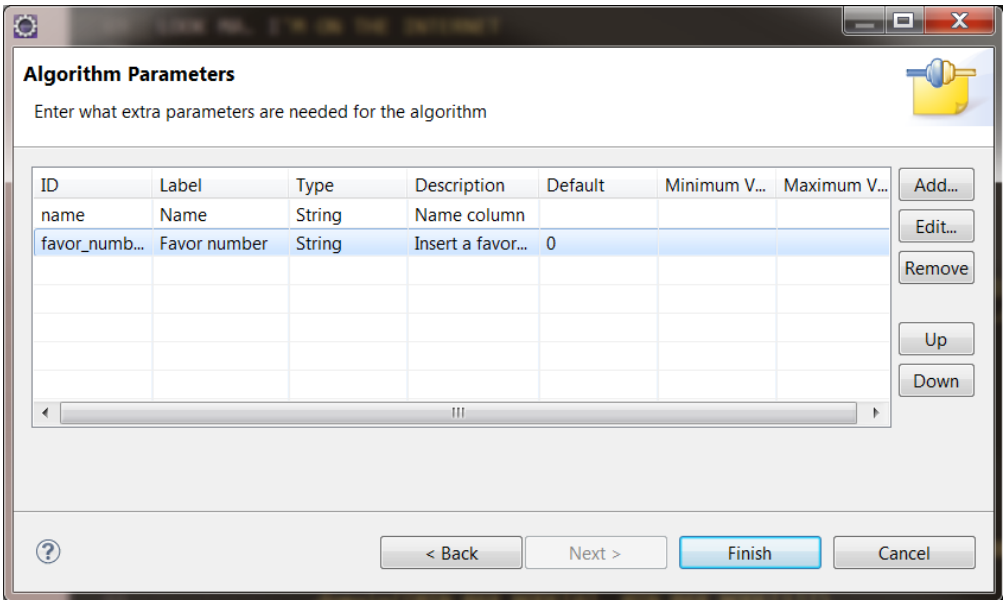
Creating an Algorithm with Tabular Data as Input

> **Note**
>
> This plugin works only with Sci2 and not with CIShell. Use the latest version of Sci2 (available at http://nwb.cns.iu.edu/nightly/sci2/).
>
> Due to a versioning issue this plugin will not work with previous versions of Sci2. To fix this, before exporting the plugin from Eclipse, change the minimum version to match of **`org.cishell.utilities`** to **`1.0.1`** in the D*ependencies* tab of *MANIFEST.MF* file.

Create a CIShell Java Algorithm Project, go to `File -> New -> Project...` and continue the step same as showed in Hello World Tutorial until the setting the input and output data step. Use **`prefuse.data.Table`** in the *Data the algorithm will take in* field as shown below.



When you are finished, click **Next >** to go to the Algorithm Parameters page.



Click the **Add** button to add a algorithm parameter and insert the following values into the input fields. You will need to add a **`name`** parameter as a String and a **`favor`** parameter as a String as in the above image.

Finally, press the **Finish** button to generate a new Java Algorithm project. Open your *algorithm.properties* file you will see a the in_data is set to `prefuse.data.Table`.

algorithm.properties

```
menu_path=AwesomeAlgorithms/additions
label=Tabular Data Practice
description=Reads data from a tabular file.
in_data=prefuse.data.Table
out_data=null
service.pid=org.my.algorithm.data.HelloWorldFromFileAlgorithm
remoteable=false
```

Mutating and Preserving Parameters

For this example we want to mutate one parameter while preserving the other. Ideally, we want to mutate the `name` field to become a drop down box containing the available column titles from the user-provided tabular data (such as a CSV file). The `favor_number` field should remain as it is. To achieve this, we will edit the factory class to implement the `ParameterMutator` interface. The factory is defined in your *HelloWorldFromFileAlgorithmFactory.java* file, which should look like this:

HelloWorldFromFileAlgorithmFactory.java

```
package org.my.algorithm.data;

import java.util.Dictionary;
import org.cishell.framework.CIShellContext;
import org.cishell.framework.algorithm.Algorithm;
import org.cishell.framework.algorithm.AlgorithmFactory;
import org.cishell.framework.algorithm.ParameterMutator;
import org.cishell.framework.data.Data;
import org.cishell.reference.service.metatype.BasicAttributeDefinition;
import org.cishell.reference.service.metatype.BasicObjectClassDefinition;
import org.cishell.utilities.TableUtilities;
import org.osgi.service.metatype.AttributeDefinition;
import org.osgi.service.metatype.ObjectClassDefinition;
import prefuse.data.Table;

public class HelloWorldFromFileAlgorithmFactory implements AlgorithmFactory, ParameterMutator {
    public Algorithm createAlgorithm(Data[] data,
                                                                    Dictionary parameters,
                                                                    CIShellContext ciShellContext) {
        return new HelloWorldFromFileAlgorithm(data, parameters, ciShellContext);
    }

    /* Mutate the parameters here */
    @Override
        public ObjectClassDefinition mutateParameters(Data[] data,
                        ObjectClassDefinition parameters) {
                Table table = (Table) data[0].getData();

                BasicObjectClassDefinition newParameters = new BasicObjectClassDefinition(
                        parameters.getID(), parameters.getName(), parameters.getDescription(), null);
                AttributeDefinition[] oldAttributeDefinitions = parameters.getAttributeDefinitions(
                        ObjectClassDefinition.ALL);
                String[] columnTitles =
                        TableUtilities.getAllColumnNames(table.getSchema()).toArray(new String[0]);
                for (AttributeDefinition oldAttributeDefinition : oldAttributeDefinitions) {
                        /*
                         * Append column titles into drop down boxes and assign to name parameter.
                         */
                        if (oldAttributeDefinition.getID().equals("name") || oldAttributeDefinition.getID().
equals("favor_number")) {
                                newParameters.addAttributeDefinition(
                                                ObjectClassDefinition.REQUIRED,
                                                new BasicAttributeDefinition(
                                                                oldAttributeDefinition.getID(),
                                                                oldAttributeDefinition.getName(),
                                                                oldAttributeDefinition.getDescription(),
                                                                oldAttributeDefinition.getType(),
                                                                columnTitles,
                                                                columnTitles));
                        } else {
                                /* None edit field */
                                newParameters.addAttributeDefinition(
                                        ObjectClassDefinition.REQUIRED, oldAttributeDefinition);
                        }
                }
                return newParameters;
        }
}
```

Now we need to edit the *HelloWorldFromFileAlgorithm.java* file to manipulate the tabular data. This time you will add the content into the `execute()` method. The first input data is always stored in `data[0]`. If there are more than one input data, you will access them through `data[0]`, `data[1]`, ... in the order defined in the *algorithm.properties* file. This example shows how could the algorithm access rows and columns through `prefuse.data.Table`.

HelloWorldFromFileAlgorithm.java

```
package org.my.algorithm.data;

import java.util.Dictionary;
import java.util.Iterator;
import org.cishell.framework.CIShellContext;
import org.cishell.framework.algorithm.Algorithm;
import org.cishell.framework.algorithm.AlgorithmExecutionException;
import org.cishell.framework.data.BasicData;
import org.cishell.framework.data.Data;
import org.cishell.framework.data.DataProperty;
import org.osgi.service.log.LogService;
import prefuse.data.Table;

public class HelloWorldFromFileAlgorithm implements Algorithm {
    private Data[] data;
    private Dictionary parameters;
    private CIShellContext ciShellContext;
        private String nameColumn;
        private String favorNumberColumn;
        private LogService logger;

    public HelloWorldFromFileAlgorithm(Data[] data,
                                       Dictionary parameters,
                                       CIShellContext ciShellContext) {
        this.data = data;
        this.parameters = parameters;
        this.ciShellContext = ciShellContext;
        this.logger = (LogService) ciShellContext.getService(LogService.class.getName());
                nameColumn = (String) parameters.get("name");
                favorNumberColumn = (String) parameters.get("favor_number");
    }
    public Data[] execute() throws AlgorithmExecutionException {
            Table inputTable = (Table) data[0].getData();
            Iterator<?> rowsIterator = inputTable.iterator();
            int nameColumnIndex = inputTable.getColumnNumber(nameColumn);
            int favorNumberColumnIndex = inputTable.getColumnNumber(favorNumberColumn);

                while (rowsIterator.hasNext()) {
                        int currentRowNumber = Integer.parseInt(rowsIterator.next().toString());
                        String name = inputTable.get(currentRowNumber, nameColumnIndex).toString();
                        Integer favorNumber;
                        try {
                                favorNumber = Integer.valueOf(inputTable.get(currentRowNumber,
favorNumberColumnIndex).toString());
                        } catch (NumberFormatException e) {
                                throw new AlgorithmExecutionException("Favor number column must hold only
integer value!", e);
                        }

                        this.logger.log(LogService.LOG_INFO, name);
                        this.logger.log(LogService.LOG_INFO, favorNumber.toString());
                }

                return null;
    }
}
```

Export the algorithm into CIShell (or Sci2). Restart the tool an you can try to run the algorithm with the *nameWithFavorNumber.csv* file as input. To load the file into the tool, you can drag and drop the CSV file into the Data Manager panel. Alternatively, you can use the `File -> Open` menu option. Then run the algorithm by selecting `AwesomeAlgorithms -> Tabular Data Practice` from the menu bar. A pop-up window will ask for the columns to be included.

Generate a Tabular Output for the Algorithm

We will continue working on the same code. You can simply edit *algorithm.properties* as follows to tell CIShell that your algorithm should generate a tabular output file.

algorithm.properties

```
menu_path=AwesomeAlgorithms/additions
label=Tabular Data Practice
description=Reads data from a tabular file.
in_data=prefuse.data.Table
out_data=prefuse.data.Table
service.pid=org.my.algorithm.data.HelloWorldFromFileAlgorithm
remoteable=false
```

Next, you will need to add the `generateOutputTable()` and `generateOutData()` into *HelloWorldFromFileAlgorithm.java* as follows:

HelloWorldFromFileAlgorithm.java

```
package org.my.algorithm.data;

import java.util.Dictionary;
import java.util.Iterator;
import org.cishell.framework.CIShellContext;
import org.cishell.framework.algorithm.Algorithm;
import org.cishell.framework.algorithm.AlgorithmExecutionException;
import org.cishell.framework.data.BasicData;
import org.cishell.framework.data.Data;
import org.cishell.framework.data.DataProperty;
import org.osgi.service.log.LogService;
import prefuse.data.Table;

public class HelloWorldFromFileAlgorithm implements Algorithm {
    private Data[] data;
    private Dictionary parameters;
    private CIShellContext ciShellContext;
        private String nameColumn;
        private String favorNumberColumn;
        private LogService logger;

    public HelloWorldFromFileAlgorithm(Data[] data,
                                       Dictionary parameters,
                                       CIShellContext ciShellContext) {
        this.data = data;
        this.parameters = parameters;
        this.ciShellContext = ciShellContext;
        this.logger = (LogService) ciShellContext.getService(LogService.class.getName());
                nameColumn = (String) parameters.get("name");
                favorNumberColumn = (String) parameters.get("favor_number");
    }
    public Data[] execute() throws AlgorithmExecutionException {
            Table inputTable = (Table) data[0].getData();
            Iterator<?> rowsIterator = inputTable.iterator();
            int nameColumnIndex = inputTable.getColumnNumber(nameColumn);
            int favorNumberColumnIndex = inputTable.getColumnNumber(favorNumberColumn);

                while (rowsIterator.hasNext()) {
                        int currentRowNumber = Integer.parseInt(rowsIterator.next().toString());
                        String name = inputTable.get(currentRowNumber, nameColumnIndex).toString();
                        Integer favorNumber;
                        try {
                                favorNumber = Integer.valueOf(inputTable.get(currentRowNumber,
favorNumberColumnIndex).toString());
                        } catch (NumberFormatException e) {
                                throw new AlgorithmExecutionException("Favor number column must hold only
integer value!", e);
                        }

                        this.logger.log(LogService.LOG_INFO, name);
                        this.logger.log(LogService.LOG_INFO, favorNumber.toString());
                }

            Table table = generateOutputTable();
```

```
        return generateOutData(table);
    }

    /* Create an output table */
    private Table generateOutputTable() {
            Table table = new Table();
            table.addColumn("name", String.class);
            table.addColumn("Favor number square root", Integer.class);
            int rowNumber = table.addRow();
            table.set(rowNumber, "name", "haha");
            table.set(rowNumber, "Favor number square root", 100);
            rowNumber = table.addRow();
            table.set(rowNumber, "name", "haha");
            table.set(rowNumber, "Favor number square root", 200);

        return table;
    }

    private Data[] generateOutData(Table table) {

                /*
                 * After getting the output in table format make it available to the user.
                 */
                Data output = new BasicData(table, Table.class.getName());
                Dictionary<String, Object> metadata = output.getMetadata();
                metadata.put(DataProperty.LABEL, "output001");
                metadata.put(DataProperty.PARENT, this.data[0]);
                metadata.put(DataProperty.TYPE, DataProperty.TABLE_TYPE);

                return new Data[] { output };
    }
}
```

The `generateOutputTable()` method shows how to create a table and add rows and columns. The `generateOutData()` shows how to create output data for return, so that Cishell's Data Manager understands how to display the output.

Export this algorithm into CIShell and run with the *name.csv* file. Now, a new tabular data set is created in the Data Manager panel. You can save by right-clicking on the target file.

To learn more about manipulate network data, see the Practical Java Algorithm Development.

Resolving Missing Dependencies

If you followed along with this example, you may notice a few errors pertaining to importing packages and missing type definitions. We can resolve these errors in the *MANIFEST.MF* file of this algorithm.

In your Algorithm directory, navigate to the *META-INF* directory. Open the *MANIFEST.MF* file. A series of tabs should now be visible under your file editor. Click *Dependencies* to see a list of all packages imported for this project. Compare these packages to what you are missing in your algorithm.

To add a package, click the **Add...** button. Type the entire name of the desired package, or part of the name surrounded by asterisks *****

Example: If you were looking for *org.cishell.reference.service.metatype*, you could search **service.metatype** and get all results which contain that sequence of characters.