Data exploration through dot-driven development (Artifact)*

Tomas Petricek[1]

1 **The Alan Turing Institute, London, UK**
and Microsoft Research, Cambridge, UK
tomas@tomasp.net

Abstract

This artifact presents The Gamma, a simple programming environment for data exploration that uses member access as the primary mechanism for constructing queries. The artifact consists of two parts. The user facing web-based component allows users to explore a simple dataset of Olympic medal winners while a back-end service provides the data and evaluates queries executed by the user.

The purpose of the artifact is to illustrate the pivot type provider, which provides a simple way for constructing queries in a object-based programming language equipped with member access. As illustrated by a number of walkthroughs, the pivot type provider can be use to construct new queries from code or using the user interface, but it also encourages the user to modify existing code.

1998 ACM Subject Classification D.3.2 Very high-level languages
Keywords and phrases Data science, type providers, pivot tables, aggregation
Digital Object Identifier 10.4230/DARTS.3.2.1
Related Conference European Conference on Object-Oriented Programming (ECOOP 2017), June 18-23, 2017, Barcelona, Spain

1 Scope

The artifact is designed to provide a repeatble way of evaluating the user experience that is provided by the pivot type provider. As discussed in the paper, the purpose of the project is to simplify data exploration (by using a simple programming language), make it open and transparent (by revealing the source code) and encourage further data exploration (by making it easy to change the source code). The presented artifact supports this by providing a number of guided walkthroughs that illustrate the above features.

The artifact web page contains a number of examples that illustrate the pivot type provider. When the page loads, the examples are executed and the results appear as illustrated in Figure 1. For each example, the web page shows the full source code using the pivot type provider together with several additional libraries not presented in the paper (such as `chart` and `table`), which are used for creating visualizations.

Clicking on the "Edit source code" button opens an editor where the code can be modified. The editor supports auto-completion, which is the key feature made possible by the type system of the language and the pivot type provider. The following three sections briefly describe three scenarios that illustrate interesting aspects of the artifact.

* This artifact is a companion of the paper: Tomas Petricek, "Data exploration through dot-driven development", Proceedings of the 31st European Conference on Object-Oriented Programming (ECOOP 2017), June 18-23, 2017, Barcelona, Spain. This work was supported in part by The Alan Turing Institute under the EPSRC grant EP/N510129/1 and by the Google Digital News Initiative.

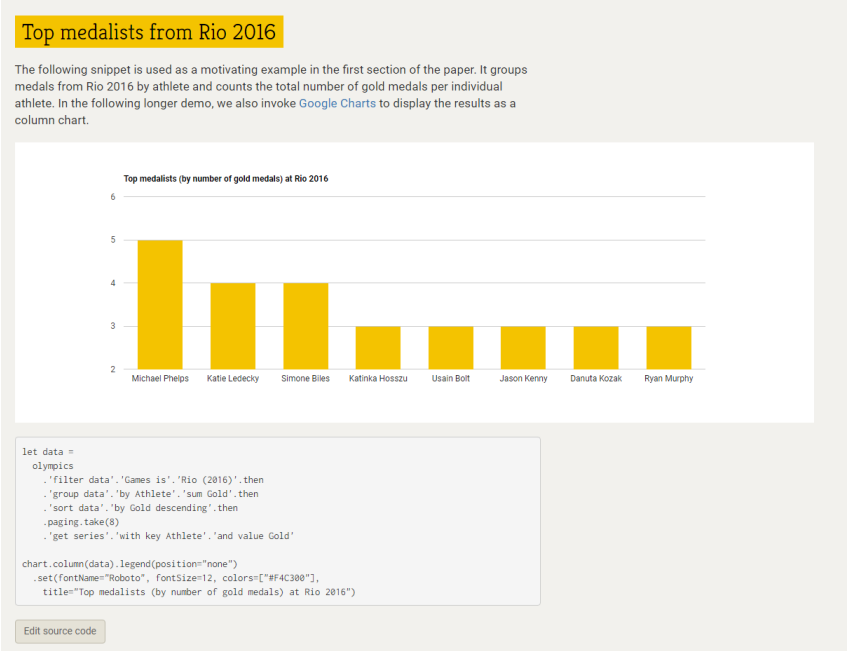

Figure 1 Visualization showing top medalist from Rio 2016 with source code

Walkthrough 1: Obtain top medallists from London 2012

In the first demo, we make a simple change to an existing source code to create a bar chart showing top medallists from London 2012 rather than top medallists from Rio 2016.

1. Go to the first visualization "Top medalists from Rio 2016" and click the "Edit source code" button to load the source code in the editor.
2. Locate the `Rio (2016)` member on the third line and delete it in the text editor. Additionally, delete the "dot" after `Games is`.
3. Type "." immediately after the quoted `Games is` member. An auto-complete list should appear showing the available Olympic games together with their years. Choose `London (2012)` and make sure the identifier is separated from the following `then` member with another "dot".
4. Click "Update page". The visualization on the original demo page should change to show the new chart.

Walkthrough 2: Obtain Mongolian gold medalists in code

This demo illustrates the experience of writing code using the pivot type provider from scratch. We start with an empty code editor, start writing code and use auto-complete to create a simple query.

1. Click on the "Edit source code" button for any of the examples and delete all source code in the editor, so that we can start from empty file.
2. Type `olympics.` and note that auto-complete offers different data transformations as documented in the paper. Start by choosing `filter data`, followed by `Team is` and `Mongolia`. Finally, complete the filtering by choosing `then`. Typing "dot" again offers the list of all available transformations.

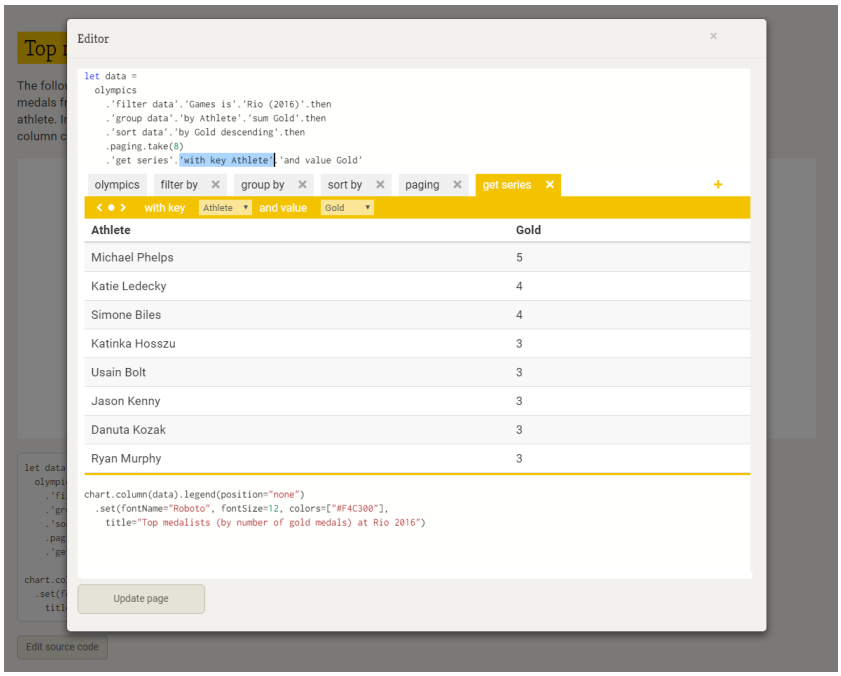

Figure 2 Visualization showing top medalist from Rio 2016 with source code.

3. Complete the query by typing "dot" and selecting items from auto-complete. When auto-complete is displayed, you can navigate using arrow keys and filter items by typing substrings of the member names. As you type, you can see the data calculated so far in the preview below the current line.

```
olympics.'filter data'.'Team is'.Mongolia.then
  .'filter data'.'Medal is'.Gold.then.'get the data'
```

4. Now that the data query is written, complete the sample by creating a table to show the results. Wrap the code in the following (note that the code written in previous step needs to be indented):

```
let altan = (...)
table.create(altan)
```

5. When you place the cursor inside the `table.create` call, you should see table as a preview. Click "Update page" and you should see the new table appear in the original demo page.

Walkthrough 3: Count medals per discipline using the pivot user interface

This demo introduces the user interface built on top of the pivot type provider as mentioned in the case study presented in the paper. We will use the user interface to construct a simple query that finds the number of medals awarded per discipline and finds the disciplines with the largest number of medals. To do this, we make small edits to the code and complete the rest of the work using a simple user interface.

1. Go to the "All time medalists table" demo and click "Edit source code".

2. In the displayed source code, click anywhere inside the query that performs the data aggregatoion so that the user interface displayed in Figure 2 appears.
3. Remove all the operations except for `group by` by clicking on the "x" button on the right. Start from the last one and remove `get the data`, `paging`, `get the data` and `sort by`. Now, remove all the calculated aggregations by clicking on "x" buttons in the yellow toolbar.
4. Change the grouping key by selecting `by Discipline` in the drop down showing `by Athlete`. Next, add aggregation `count all` by clicking on the "+" button in the yellow toolbar.
5. Add `sort by` transformation by clicking on the yellow "+" in the tab panel showing a list of operations. Then add sorting key `by count descending` by clicking on the "+" in the yellow toolbar.
6. Finally, add the `get the data` transformation to obtain the data. The final source code should look as follows:

```
let data =
  olympics
    .'group data'.'by Discipline'.'count all'.then
    .'sort data'.'by count descending'.then
    .'get the data'
table.create(data)
```

7. Finally, click "Update page" and review the newly produced table in the main page. You should see athletics at the top with 3856 medals and Jeu de Paume at the bottom with just 3 medals.

2 Content

The artifact package includes:

- a Docker image (`thegamma-ecoop17.tar`), which contains the necessary client-side and server-side components for running the artifact
- an F# source code (`thegamma-source.zip`) which contains the full source code for the client-side and server-side components behind the artifact

To make evaluating of the user interface easier, we provide a Docker container that can be used to host the system. The Docker container is based on the standard `fsharp` image supplied by the F# Software Foundation.

The included source code provides complete source code for the server-side components (running as F# code on the server) and client-side components (translated to JavaScript using the Fable compiler). When compiling code from scratch, several packages are downloaded prior to the compilation from standard .NET and JavaScript package repositories NuGet and npm.

3 Getting the artifact

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS).

Running the artifact

The presented artifact consists of a simple web server packaged as a Docker container. When started, the web server hosts the service that provides data to the web browser, as well as a sample web page with a number of demos illustrating the language.

Running the artifact requires Docker (`https://www.docker.com`) and the Google Chrome browser (`http://www.google.co.uk/chrome`) and also download the provided docker image

`thegamma-ecoop17.tar`. To run the artifact, use the `docker load` command to unpack the image and `docker run` to start the web server:

```
docker load --input thegamma-ecoop17.tar
docker run -p 8889:80 thegamma-ecoop17
```

Alternatively, the `thegamma-ecoop17` Docker image is also hosted on the Docker repository and can be downloaded and executed directly from the Docker repository:

```
docker run -p 8889:80 tomasp/thegamma-ecoop17
```

The above commands specify the `-p 8889:80` parameter, which instructs Docker to map the server hosted at port 80 inside the Docker container to a port 8889 of the host machine. Once the Docker container starts, you will be able to access the sample web site by opening `http://localhost:8889` in the Google Chrome browser.

Latest version

The tool implemented as part of the submitted paper is also available as a JavaScript library that is made available through the standard JavaScript package manager npm. The full project soruce code is also hosted on GitHub. The following links can be used to obtain the latest version:

- `https://github.com/the-gamma/thegamma-script`
- `https://npmjs.com/package/thegamma-script`

4 Tested platforms

The artifact is known to work on any platform running Docker version 17. It has been specifically tested using Docker version 17.03.1-ce-win12 on the Windows operating system. 5 GB of free space on disk and at least 2 GB of free space in RAM are sufficient for running the artifact.

5 License

MIT (https://opensource.org/licenses/MIT)

6 MD5 sum of the artifact

0e0e2873157bb6d69cbff5822523e41a

7 Size of the artifact

335 MB