Understanding RDataTracker's Prov-Json Output

json ver. 2.1

1. The Provenance Graph

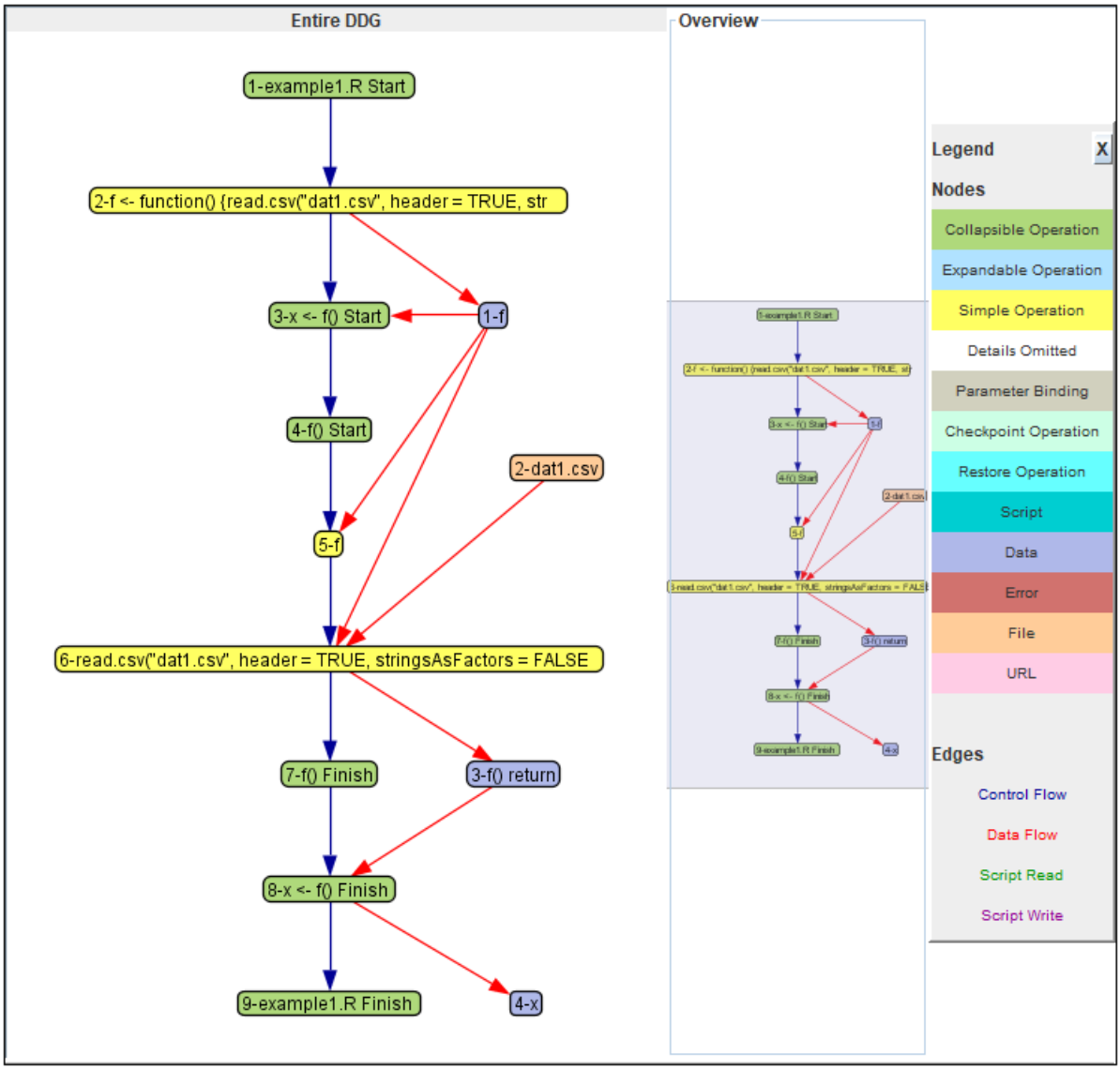
Our provenance-collection tools represents the provenance collected from an execution of an R script in the form of a directed graph..

This is an example of a script which could be run with our tools:

```
f <- function()
{
    read.csv( "dat1.csv", header = TRUE , stringsAsFactors = FALSE )
}

x <- f()
```

This is the graph produced, visualised using DDGExplorer:



There are a few different types of nodes, as indicated by their different colours:

- Collapsible and Expandable Operations (green and light blue): These represent Start and Finish procedure nodes used in nesting and grouping of operations.
- Simple Procedure nodes (yellow): An executed section of code. Normally a statement or line of code.
- Details Omitted (white): These represent parts in the code where provenance was not collected.
- Parameter binding (grey): These are when values are bound to a parameter when used by a function.
- Data nodes (purple): Data that is produced by or used by a procedure. Normally in the form of data assigned to a variable.
- Error (red): A .type of data node representing a warning or exception thrown by a statement.
- File (orange): A file read in or written to by the script.
- URL (pink): A link to an external source where data is obtained from.

Different types of edges connect the different nodes in the graph:

- Procedure-to-procedure (control flow): Links procedure nodes to indicate control flow.
- Procedure-to-data (data output): Links a procedure node to a data node to indicate data produced.
- Data-to-procedure (data input): Links a data node to a procedure node to indicate data used.

2. The JSON file

This graph, the output of our provenance-collection tool, is stored as a JSON file. Our JSON format follows and extends the W3 PROV-JSON standard, abiding by its schema as well as defining conventions of our own in order to better express provenance generated by our tools.

The overall skeleton shown below:

```
{
    "prefix": {
        // prefixes used in this prov-json file
    },

    "agent" : {
        // agent: this json file and the tool that produced this
    },

    "activity" : {
        // procedure nodes (p)
    },

    "entity" : {
        // data nodes
        // environment
        // library nodes (l)
        // function nodes (f)
    },

    "wasInformedBy" : {
        // procedure-to-procedure edges (pp)
    },

    "wasGeneratedBy" : {
        // procedure-to-data edges (pd)
    },

    "used" : {
        // data-to-procedure edges (dp)
        // function-to-procedure edges (fp)
    },

    "hadMember" : {
        // groups function nodes with their library nodes
    }
}
```

These sections are defined by the PROV-JSON standard. To fit our needs, we expressed each node and edge of the data-derivation graph as json objects in these sections. As there are more types of nodes and edges than there are sections, sections such as “entity” and “used” have more than one type of node or edge. This customisation is further explained below:

a. prefix

Attributes within this section defines the namespaces (prefixes) used in the file. Each keyword is preceded by the prefix which indicates the namespace it is from.

Prefixes used:

- prov: The prov namespace defined by W3 containing a set of reserved keywords.
- rdt: The set of keywords defined by our prov-json extension.

The xsd prefix is also used, but it is implicitly defined by the prov-json standard.

```
"prefix": {
    "prov": "http://www.w3.org/ns/prov#",
    "rdt": "http://rdatatracker.org/"
},
```

b. agent

The PROV data model defines an agent as follows:
"An agent is something that bears some form of responsibility for an activity taking place, for the existence of an entity, or for another agent's activity."

We adapted this to contain a node with the following information:

- The name of the tool which produced the json file.
- The version number of that tool.
- The version number of our prov-json representation.

```
"agent" : {
    // agent: this json file and the tool that produced this
    "rdt:a1": {
        "rdt:tool.name": "RDataTracker",
        "rdt:tool.version": "2.27.0",
        "rdt:json.version": "2.1"
    }
},
```

c. activity

The PROV data model defines an activity as follows:
"An activity is something that occurs over a period of time and acts upon or with entities; it may include consuming, processing, transforming, modifying, relocating, using, or generating entities."

This is the section is where our procedure nodes are. Each node is numbered in the order they were executed.

Each node has the following fields:

- name: The name of the node. In most cases, it is the procedure itself or a shortened version of it.
- type - Operation, Start, Finish, Binding
    - Operation: An executed statement or procedure. The most common type of procedure node.
    - Start/Finish: Nodes that wrap around a block of procedures. An example of this is a function call.
    - Binding: Parameter binding.
- elapsedTime: The time this procedure took to execute.
- scriptNum: The script number which this procedure is from.
- startLine, startCol, endLine, endCol: The line and column numbers the procedure starts and ends with

Below is the json representation of a few activity nodes from the json file generated when running RDataTracker on the very first example in this document:


```
"rdt:p2": {
    "rdt:name": "f <- function() {\tread.csv(\"dat1.csv\", header = TRUE, str",
    "rdt:type": "Operation",
    "rdt:elapsedTime": 1.64,
    "rdt:scriptNum": 0,
    "rdt:startLine": 1,
    "rdt:startCol": 1,
    "rdt:endLine": 4,
    "rdt:endCol": 1
},
"rdt:p3": {
    "rdt:name": "x <- f()",
    "rdt:type": "Start",
    "rdt:elapsedTime": 1.64,
    "rdt:scriptNum": 0,
    "rdt:startLine": 6,
    "rdt:startCol": 1,
    "rdt:endLine": 6,
    "rdt:endCol": 8
},
```


d. entity

The PROV data model defines an entity as follows:
"An entity is a physical, digital, conceptual, or other kind of thing with some fixed aspects; entities may be real or imaginary."
We use this section for data nodes, the environment, library nodes, and function nodes.

Data nodes contain the following fields:

- name: The name of the node. In most cases, this is the variable name.
- value: The value of the data. For large values such as lists and data frames, the data are stored as snapshots, normally either as R objects or csv files. This value will then be the path to that snapshot. If in the case where snapshots are not being saved (by setting the max.snapshot.size parameter to 0 when calling prov.run), this value will be "NotRecorded".

- valType: This is a json-object expressed as a string as the Prov-Json standard does not allow json objects at this level of nesting. It contains the following fields:
    - container: The container type. Examples are vector, matrix, and data frame.
    - dimension: The size of the object.
    - type: The types contained within the container. In containers such as lists and data frames where the types may differ between elements, this will list the type of each element (in lists) or the type of each column (for data frames).
- type: The type of data. The different values it can be are:
    - Data: The most common type. This is data stored in memory. This is also the type of data node for snapshots whose values are not saved.
    - Snapshot: Data that is larger than a predetermined size are stored as snapshots.
    - File: A file.
    - URL: A url.
    - Exception, Warning: Exceptions and warnings thrown by a procedure.
- scope: The scope, or environment, where this data resides in.
- fromEnv: This value is set to TRUE if the variable was set in the Global Environment before the script was run, FALSE otherwise.
- hash: The hash number for this node if the data node is for a file.
- timestamp: If applicable, this records the timestamp of the original file.
- location: If applicable, this records the name and path of the original file.

Below is the json representation of the data nodes from the json file generated when running RDataTracker on the very first example in this document:


```
"rdt:d2": {
    "rdt:name": "dat1.csv",
    "rdt:value": "data/2-dat1.csv",
    "rdt:valType": "{\"container\":\"vector\", \"dimension\":[1], \"type\":[\"character\"]}",
    "rdt:type": "File",
    "rdt:scope": "undefined",
    "rdt:fromEnv": false,
    "rdt:hash": "319b41646609480201 24f58b815c4a10",
    "rdt:timestamp": "2018-07-17T09.47.44EDT",
    "rdt:location": "C:/Users/fong22e/Documents/provjson/examples/dat1.csv"
},
"rdt:d3": {
    "rdt:name": "f() return",
    "rdt:value": "data/3-f() return.csv",
    "rdt:valType": "{\"container\":\"data_frame\", \"dimension\":[5,3], \"type\":[\"integer\",\"character\",\"logical\"]}",
    "rdt:type": "Snapshot",
    "rdt:scope": "R_GlobalEnv",
    "rdt:fromEnv": false,
    "rdt:hash": "",
    "rdt:timestamp": "2018-07-17T09.47.44EDT",
    "rdt:location": ""
},
"rdt:d4": {
    "rdt:name": "x",
    "rdt:value": "data/4-x.csv",
    "rdt:valType": "{\"container\":\"data_frame\", \"dimension\":[5,3], \"type\":[\"integer\",\"character\",\"logical\"]}",
    "rdt:type": "Snapshot",
    "rdt:scope": "R_GlobalEnv",
    "rdt:fromEnv": false,
    "rdt:hash": "",
    "rdt:timestamp": "2018-07-17T09.47.44EDT",
    "rdt:location": ""
},
```

The **environment** node contains information about the execution environment. It contains the following fields:

- name: Contains the value "environment". Shows that this is the environment node.
- architecture: The computer"s architecture.
- operatingSystem: The operating system.
- language: The language the script is in.
- langVersion: The version of the language the script is in.
- script: The full path of the executed script.
- scriptTimeStamp: The timestamp of the executed script.
- sourcedScripts: If the source function is called, this stores the names of the scripts sourced in a list.
- sourcedScriptTimeStamps: The timestamps of scripts run using the source function, if any. This list has the same length as sourcedScripts. We initially tried to express this information as a list of sourcedScript and timestamp pairs, but the prov-json standard does not allow json objects at that level of nesting.
- workingDirectory: The directory where the script was executed.
- ddgDirectory: The directory where the json file is stored.
- ddgTimeStamp: The timestamp of the json file.
- hashAlgorithm: The name of the algorithm used to produce hash values for nodes.

Below is the environment node from the json file generated when running RDataTracker on the very first example in this document:


```
"rdt:environment": {
    "rdt:name": "environment",
    "rdt:architecture": "x86_64",
    "rdt:operatingSystem": "windows",
    "rdt:language": "R",
    "rdt:langVersion": "R version 3.5.0 (2018-04-23)",
    "rdt:script": "C:/Users/fong22e/Documents/provjson/examples/example1.R",
    "rdt:scriptTimeStamp": "2018-07-17T09.39.51EDT",
    "rdt:sourcedScripts": "",
    "rdt:sourcedScriptTimeStamps": "",
    "rdt:workingDirectory": "C:/Users/fong22e/Documents/provjson/examples",
    "rdt:ddgDirectory": "getwd/prov_example1",
    "rdt:ddgTimeStamp": "2018-07-17T09.47.44EDT",
    "rdt:hashAlgorithm": "md5"
},
```

This is another example where the script had called the R source function:

```
"rdt:environment": {
    "rdt:name": "environment",
    "rdt:architecture": "x86_64",
    "rdt:operatingSystem": "windows",
    "rdt:language": "R",
    "rdt:rVersion": "R version 3.5.0 (2018-04-23)",
    "rdt:script": "C:/Users/fong22e/Documents/provjson/examples/example2.R",
    "rdt:scriptTimeStamp": "2018-06-22T13.23.04EDT",
    "rdt:sourcedScripts": [
        "file1.R",
        "file2.R",
        "file3.R"
    ],
    "rdt:sourcedScriptTimeStamps": [
        "2018-06-22T12.07.45EDT",
        "2018-06-22T12.07.58EDT",
        "2018-06-22T12.08.09EDT"
    ],
    "rdt:workingDirectory": "C:/Users/fong22e/Documents/provjson/examples",
    "rdt:ddgDirectory": "./example2_ddg",
    "rdt:ddgTimeStamp": "2018-06-22T13.30.34EDT",
    "rdt:rdatatrackerVersion": "2.27.0",
    "rdt:hashAlgorithm": "md5"
},
```

```
print("example 2")

source("file1.R")
source("file2.R")
source("file3.R")
```

A **library** node is a node representing a library or package loaded at the time of execution. It contains the following fields:

- name: The name of the loaded library.
- version: The version number of the library.
- prov:type: This indicates that this node is a collection. This is used in grouping function nodes with their library nodes in the hasMember section.

```
"rdt:l1": {
    "name": "base",
    "version": "3.5.0",
    "prov:type": {
        "$": "prov:Collection",
        "type": "xsd:QName"
    }
}
```

A **function** node is a node representing a function called from a library that is neither user-defined, nor from the base R library. It contains one field showing the name of the function. Functions originating from the same library are grouped together using edges in the hadMember section, described in 2f below.

```
"rdt:f1": {
    "name": "as.roman"
}
```

Sometimes, multiple libraries will have functions with the same name. In this case, there will be multiple functions with the same name attribute. These are grouped with their respective libraries with edges in the hadMember section. For example:

```
"hadMember": {
    // groups function nodes with their library nodes
    "rdt:m1": {
        "prov:collection": "rdt:l8",
        "prov:entity": "rdt:f1"
    },
    "rdt:m2": {
        "prov:collection": "rdt:l9",
        "prov:entity": "rdt:f2"
    }
}
```

```
"rdt:f1": {
    "name": "a.function"
},
"rdt:f2": {
    "name": "a.function"
}
```

e. wasInformedBy

This section represents the concept of communication, where an activity is related to another activity. We use this section for procedure-to-procedure edges as they denote control flow moving from a procedure to the next, from the informant to the informed.

```
"rdt:pp1": {
    "prov:informant": "rdt:p1",
    "prov:informed": "rdt:p2"
},
"rdt:pp2": {
    "prov:informant": "rdt:p2",
    "prov:informed": "rdt:p3"
},
```

f. wasGeneratedBy

This section represents the concept of generation, where an activity produces an entity. We use this section for procedure-to-data (data output) edges.

```
"wasGeneratedBy" : {
    // procedure-to-data edges
    "rdt:pd1": {
        "prov:activity": "rdt:p2",
        "prov:entity": "rdt:d1"
    },
    "rdt:pd2": {
        "prov:activity": "rdt:p5",
        "prov:entity": "rdt:d2"
    },
    "rdt:pd3": {
        "prov:activity": "rdt:p7",
        "prov:entity": "rdt:d3"
    }
},
```

g. used

The opposite of generation, this section represents the concept usage, where an activity utilises an entity. We use this section for data-to-procedure (data input) edges, as well as function-to-procedure (function usage) edges.

```
"rdt:dp1": {
    "prov:entity": "rdt:d1",
    "prov:activity": "rdt:p3"
},
```

An example of a data-to-procedure edge.

```
"rdt:fp1": {
    "prov:entity": "rdt:f1",
    "prov:activity": "rdt:p7"
}
```

An example of a function-to-procedure edge.

h. hadMember

This represents the concept of membership, grouping each member entity with their respective collection entity. We use this section to group each function node with their library node, indicating which library a function is from.

```
"hadMember" : {
    // groups function nodes with their library nodes
    "rdt:m1": {
        "prov:collection": "rdt:l9",
        "prov:entity": "rdt:f1"
    }
}
```