# WGL – A Workflow Generator Language and Utility

Technical Report

Luiz Meyer, Marta Mattoso, Mike Wilde, Ian Foster

## Introduction

Many scientific applications can be characterized as having sets of input and derived data that have to be processed in several steps by a set of programs. Data transfer between these steps is usually done via files. Consequently, DAGs have been used to model these applications, and cluster and grid environments are used to process them.

Several projects have focused on the development of techniques for scheduling and executing workflows expressed as DAGs on Grids. The goal of the **wgl** tool and its related utility, **genwf** is to enable users to generate workflows that can be used to evaluate such techniques with different workloads and processing approaches.

WGL enables the modeling of DAGs with different sizes and shapes, from very small to very large. DAG specification is done by defining in a parameter file the set of datasets and programs that form the DAG. The **genwf** command reads the parameter file and produces what we call a "dax", that is, a "DAG in a XML format". **Genwf** uses VDS commands (**vdlt2vdlx**, **updatevdc**, **gendax**) to generate the dax file. In addition to the dax, **genwf** also creates all the necessary input files for processing the workflow. Figure 1 depicts the processing flow of generating a runnable workflow via **wgl** and **genwf**.

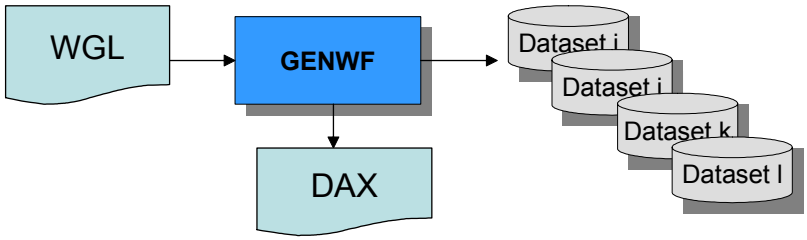



Fig. 1. Genwf functionality

## WGL Statements

**1. dataset statement: dataset( name, # files, size-unit )**

A dataset represent a set of files. Each dataset is modeled using three parameters: the dataset name, the number of files in the dataset, and the size of each file. The size can be expressed in three units: K (kilobytes), M (megabytes) and G (gigabytes). The files in a dataset are named by appending a sequence number to the name of the dataset.

**2. program statement: program(name, input_datasets, output_datasets, #instances, sleep time)**

Each program is specified using five parameters: program name, list of input datasets, list of output datasets, number of instances and the duration of the program (simulated by sleeping). The list of input datasets and output datasets must be specified as a comma-separated list between square brackets "[ ]".

The relationship between datasets and programs is modeled according to the following rules:

- **SISO** (Single Input Single Output):
  - One instance of a program takes as input a dataset with a single file and produces a dataset with a single file;
  - Each instance of a program takes as input a file from an input dataset and produces a file in the output dataset.

- **SIMO** (Single Input Multiple Output):
  - One instance of a program takes as input the file from the input dataset and produces multiple files in the output dataset;
  - Each instance of a program takes as input the file from the input dataset and produces one file in the output dataset.

- **MISO** (Multiple Input Single Output):
  - One instance of a program takes as input a dataset with multiple files and produces a dataset with one file.

- **MIMO** (Multiple Input Multiple Output):
  - One instance of a program takes as input a dataset with multiple files and produces a dataset with multiple files;
  - Each instance of a program takes as input a subset of files from the input dataset and produces one file in the output dataset;
  - Each instance of a program takes as input a subset of files from the input dataset and produces a subset of files in the output dataset;
  - Each instance of a program takes as input all files from the input dataset and produces one file in the output dataset.

Figure 2 illustrates the relationship models while table1 shows the WGL statements to model the respective relationship.

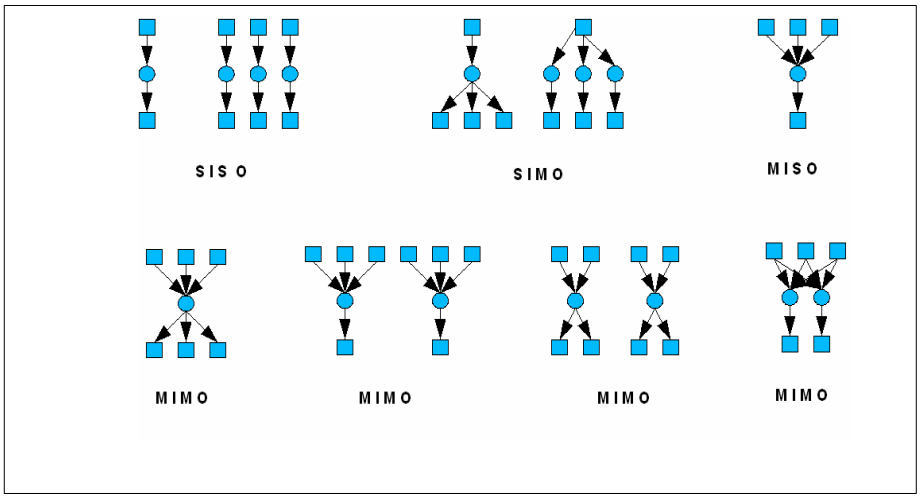



**Fig. 2.** Program-to-Dataset relationships

**Table 1.** WGL statements to model Program-to-Dataset relationships

| | |
|---|---|
| **SISO**<br>a) dataset(F1:1:10m)<br><br>dataset(F2:1:10m)<br>program(P1:F1:F2:1:30)<br><br>b) dataset(F1:3:10m)<br>dataset(F2:3:10m)<br>program(P1:F1:F2:3:30)<br><br>**SIMO**<br>a) dataset(F1:1:10m)<br>dataset(F2:3:10m)<br>program(P1:F1:F2:1:30)<br><br>b) dataset(F1:1:10m)<br>dataset(F2:3:10m)<br>program(P1:F1:F2:3:30)<br><br>**MISO**<br>a) dataset(F1:3:10m)<br>dataset(F2:1:10m)<br>program(P1:F1:F2:1:30) | **MIMO**<br>a) dataset(F1:3:10m)<br>dataset(F2:3:10m)<br>program(P1:F1:F2:1:30)<br><br>b) dataset(F1:6:10m)<br>dataset(F2:2:10m)<br>program(P1:F1:F2:2:30)<br><br>c) dataset(F1:4:10m)<br>dataset(F2:4:10m)<br>program(P1:F1:F2:2:30)<br><br>d) dataset(F1:3:10m)<br>dataset(F2:2:10m)<br>program(P1:F1*:F2:2:30) |

## Composing a Benchmark Suite with WGL

The wgl/genwf tools enable the definition of a large number of DAGS with different shapes, sizes and execution times. We define an initial set of four workflows that present different characteristics which are usually found in scientific workflows: pipeline of programs, merge of results, parallel execution of components, files being

processed by different programs. These characteristics can be shaped with parameters in order to generate DAGS that consume small or large amounts of data, producing large or small datasets and being executed by fast or slow programs. The main metrics that are intended to be measured are: execution elapse time, number of file transfers and storage utilization. The metrics can be evaluated according to different algorithms for data placement and scheduling of jobs.

**WorkFlow1:** A set of input data to be processed by one program (P2). This processing can be done in parallel by partitioning the input data (F1). In order to run, P2 makes use of a reference database (F3). Program P3 is responsible for gathering the results from P2 execution and producing the final result. Figures 4 and 5, respectively, illustrate the schema and the DAG for workflow1.

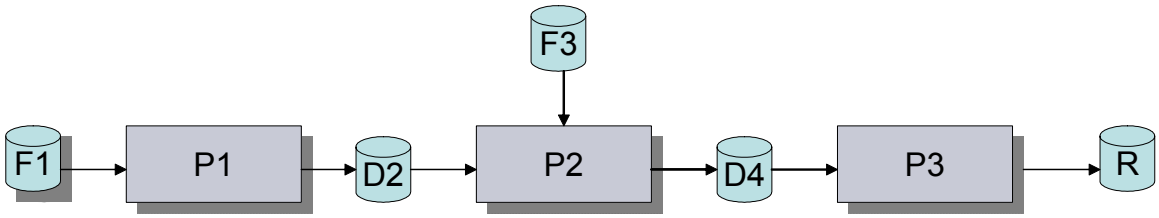



**Fig. 3.** Workflow1 scheme.

The input data can be partitioned in a number of fragments that are processed by each instance of the program independently. The result DAG will have the following shape.

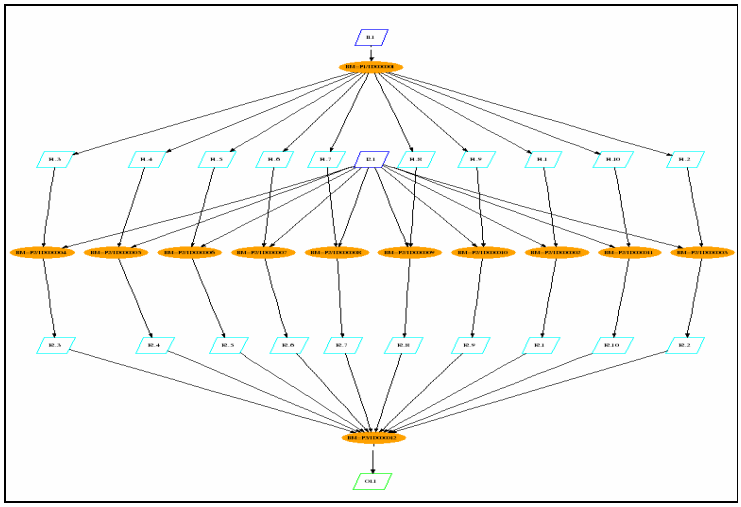


**Fig. 4.** Dag shape for workflow1

**WGL:**

*dataset(F1:1:10m)*
*dataset(F2:10:10m)*
*dataset(F3:1:1m)*
*dataset(F4:10:1m)*
*dataset(R:1:20m)*
*program(P1:F1:F2:1:30)*
*program(P2:[F2,F3]:F4:10:30)*
*program(P3:F4:R:1:20)*

**Workflow2:** Two different sets of input files have to be processed by the workflow composed by a pipeline of programs and a final merge. One file of each set of input data is processed by each instance of the first program in the pipeline part of the WF. Figure 5 illustrates the workflow2 schema while figure 6 shows the shape of the corresponding DAG.

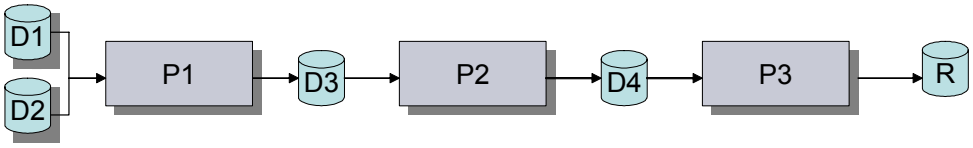
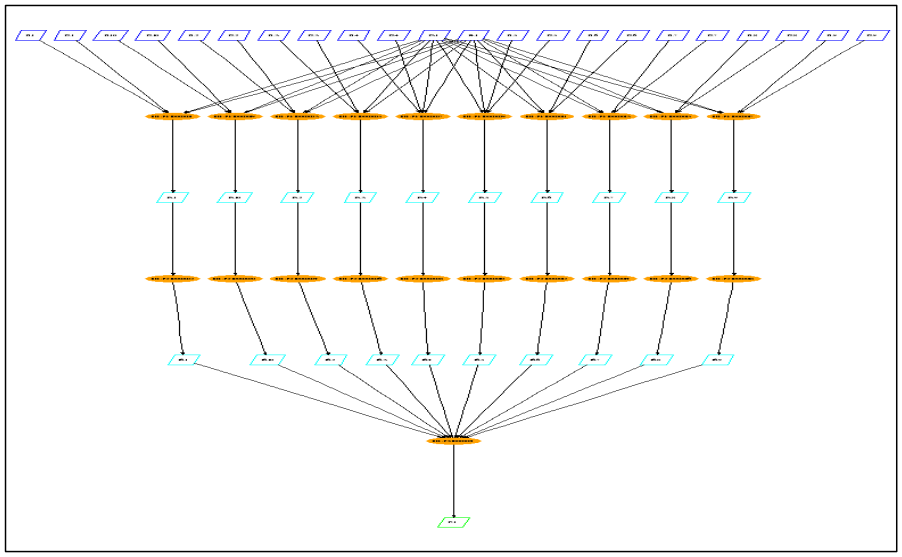



**Fig. 5.** Workflow2 scheme



**b**

**WGL:**

*dataset(F1:10:100k)*
*dataset(F2:10:1k)*
*dataset(F3:10:30k)*
*dataset(F4:10:15k)*
*dataset(R:1:20k)*
*program(P1:[F1,F2]:F3:1:300)*
*program(P2:F3:F4:10:20)*
*program(P3:F4:R:1:20)*

**Workflow3:** A set of jobs has to process a set of files. However, the files produced in the first stage for one dataset are processed by many subsequent jobs in the workflow. Figure 7 illustrates the scheme of the third workflow and figure 8 illustrates the corresponding DAG shape.

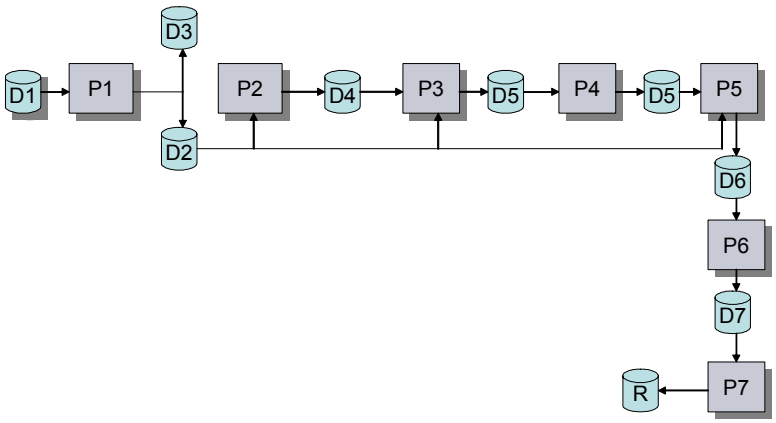



**Fig. 6.** Workflow 3 scheme

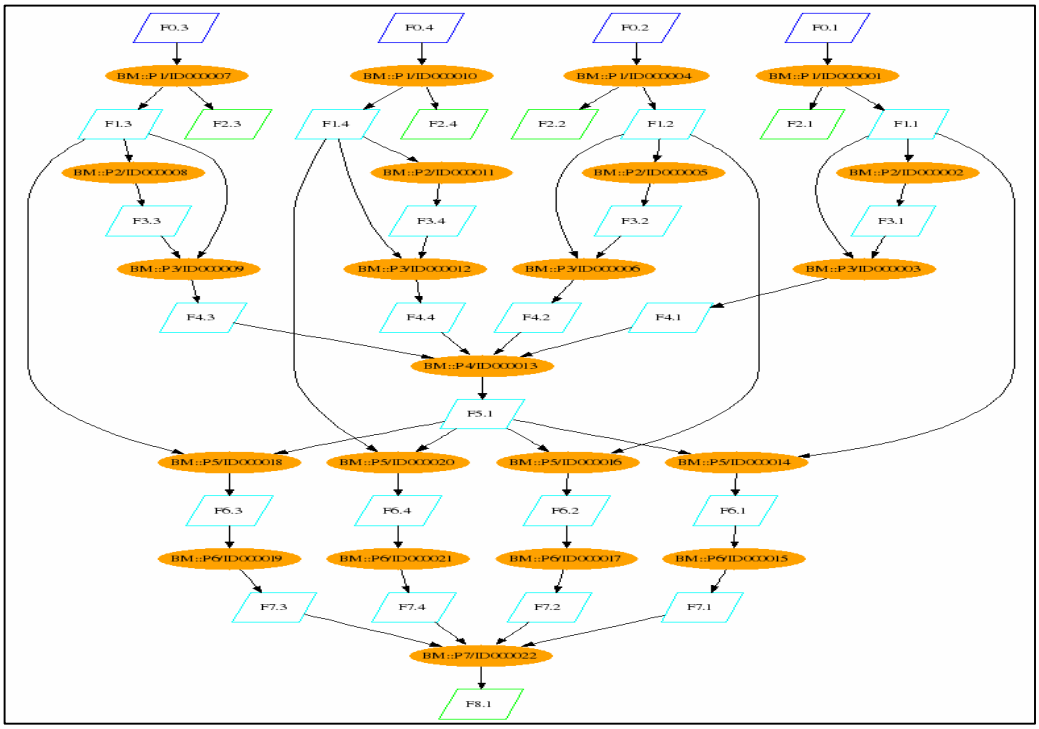



**Fig. 7.** Dag's shape for workflow 3

**WGL:**

*dataset(F1:4:100k)*
*dataset(F2:4:1k)*
*dataset(F3:4:30k)*
*dataset(F4:4:15k)*
*dataset(F5:4:15k)*
*dataset(F6:1:15k)*
*dataset(F7:4:1k)*
*dataset(F8:4:1k)*
*dataset(R:1:20k)*
*program(P1:F1:[F2,F3]:4:300)*
*program(P2:F2:F4:4:20)*
*program(P3:[F2,F4]:F5:4:20)*
*program(P4:F5:F6:1:30)*
*program(P5:[F2,F6]:F7:4:30)*
*program(P6:F7:F8:4:50)*
*program(P7:F8:R:1:30)*

**Workflow4**: A set of input files have to be processed by the workflow. The first and the second programs execute in a pipeline while the third and the fourth programs can run simultaneously after the second one. Figure 9 shows the workflow4 schema and figure 10 illustrates its DAG.

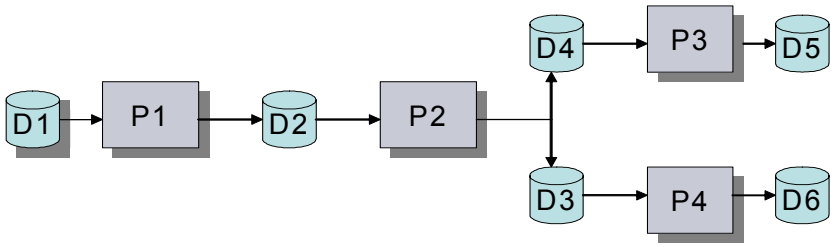



Fig. 8. Workflow 4 schema

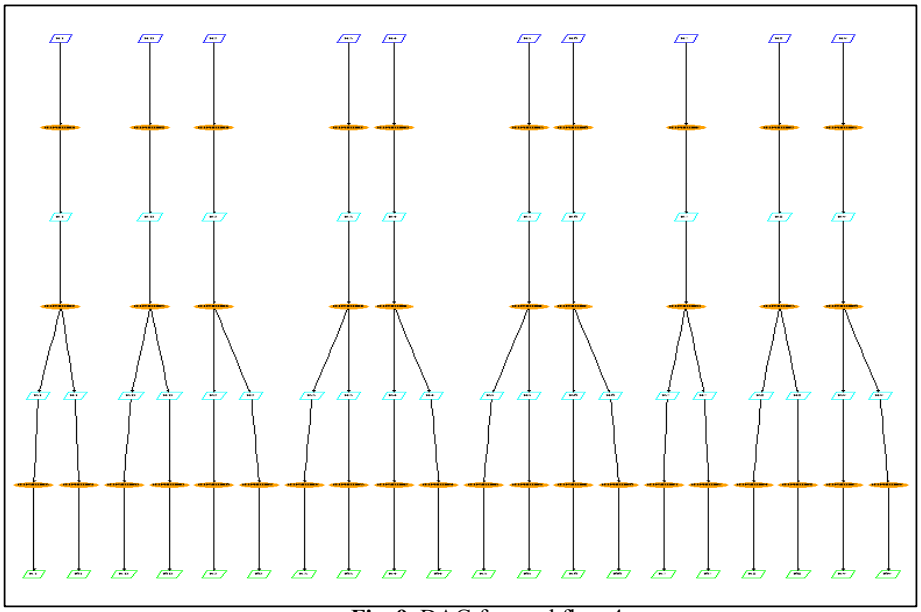


**Fig. 9.** DAG for workflow 4

**WGL:**

```
dataset(F1:10:1k)
dataset(F2:10:1k)
dataset(F3:10:30k)
dataset(F4:10:15k)
dataset(R1:10:20k)
dataset(R2:10:20k)
program(P1:F1:F2:10:300)
program(P2:F2:[F3,F4]:10:20)
program(P3:F3:R1:10:20)
program(P4:F4:R2:10:30)
```

## Downloading and Usage

A preliminary version of **WGL/GENWF** is available for test. It can be downloaded from the following address:
http://evitable.uchicago.edu/twiki/bin/view/VDSDevelopment/WorkingDocumentsb

### How to use WGL/GENWF

1) Untar the tar ball.
2) Edit a file with the WGL statements (parameter file) for generating the workflow.
3) Run the **genwf** command:

./genwf.sh –p *parameter-file* –o *vdlfile*

4) After running genwf.sh, 4 files will be created in ../wgl directory:
   - *vdlfile.vdl*
   - *vdlfile.xml*
   - *vdlfile.dax*
   - *dvlist – (contains the list of derivations in the workflow).*

Also, the input files needed to run the workflow will be created in ../wgl/data. The file *vdsfile.dax* is ready to be used as input to Euryale and Pegasus planners.

## Appendix: VDL generated for the examples

### Example 1:

```
TR BM::P1(
input F1,
output F2[])
  {
  argument = "-T 30 -G 10485760 ";
  argument = "-i "${input:F1};
  argument = "-o "${output:F2};
  }
DV BM::dv1->BM::P1(
F1=@{input:"F1.1"},    F2=[@{output:"F2.1"},@{output:"F2.2"},@{output:"F2.3"},@{output:"F2.4"},
@{output:"F2.5"},@{output:"F2.6"},@{output:"F2.7"},@{output:"F2.8"},@{output:"F2.9"},@{output:"F2.10"}]
);
TR BM::P2(
input F2,
input F3,
output F4)
  {
  argument = "-T 30 -G 10485760 1048576 ";
  argument = "-i "${input:F2};
  argument = "-i "${input:F3};
  argument = "-o "${output:F4};
  }
DV BM::dv2->BM::P2 (
    F2=@{input:"F2.1"},
    F3=@{input:"F3.1"},
    F4=@{output:"F4.1"}
);
DV BM::dv3->BM::P2(
    F2=@{input:"F2.2"},
    F3=@{input:"F3.1"},
    F4=@{output:"F4.2"}
);
. . .
TR BM::P3 (
input F4[],
output R)
  {
  argument = "-T 20 -G 10485760 1048576 20971520 ";
  argument = "-i "${input:F4};
  argument = "-o "${output:R};
  }
DV BM::dv12->BM::P3 (

F4=[@{input:"F4.1"},@{input:"F4.2"},@{input:"F4.3"},@{input:"F4.4"},@{input:"F4.5"},@{input:"F4.6"},@{input:"F4.7"},@{input:"F4.8"},
@{input:"F4.9"},@{input:"F4.10"}],
      R=@{output:"R.1"}
);
```

## Example 2:

```
TR BM::P1(
input F1[],
input F2[],
output F3[])
  {
  argument = "-T 300 -G 30720 ";
  argument = "-i "${input:F1};
  argument = "-i "${input:F2};
  argument = "-o "${output:F3};
  }
  DVBM::dv1->BM::P1(
  F1=[@{input:"F1.1"},@{input:"F1.2"},@{input:"F1.3"},@{input:"F1.4"},@{input:"F1.5"},@{input:"F1.6"},@{input:"F1.7"},@{input:"F1.
  8"},@{input:"F1.9"},@{input:"F1.10"}],
  F2=[@{input:"F2.1"},@{input:"F2.2"},@{input:"F2.3"},@{input:"F2.4"},@{input:"F2.5"},@{input:"F2.6"},@{input:"F2.7"},@{input:"F2.
  8"},@{input:"F2.9"},@{input:"F2.10"}],
  F3=[@{output:"F3.1"},@{output:"F3.2"},@{output:"F3.3"},@{output:"F3.4"},@{output:"F3.5"},@{output:"F3.6"},@{output:"F3.7"},@{o
  utput:"F3.8"},@{output:"F3.9"},@{output:"F3.10"}]
);
TR BM::P2(
input F3,
output F4)
  {
  argument = "-T 20 -G 30720 15360 ";
  argument = "-i "${input:F3};
  argument = "-o "${output:F4};
  }
DV BM::dv2->BM::P2(
    F3=@{input:"F3.1"},
    F4=@{output:"F4.1"}
);
DV BM::dv3->BM::P2(
    F3=@{input:"F3.2"},
    F4=@{output:"F4.2"}
);
…
TR BM::P3(
input F4[],
output R)
  {
  argument = "-T 20 -G 30720 15360 20480 ";
  argument = "-i "${input:F4};
  argument = "-o "${output:R};
  }
DV BM::dv12->BM::P3(

  F4=[@{input:"F4.1"},@{input:"F4.2"},@{input:"F4.3"},@{input:"F4.4"},@{input:"F4.5"},@{input:"F4.6"},@{input:"F4.7"},@{input:"F4.
  8"},@{input:"F4.9"},@{input:"F4.10"}],
    R=@{output:"R.1"}
);
```

### Example 3:

```
TR BM::P1(
input F1,
output F2,
output F3)
  {
  argument = "-T 300 -G 1024 30720 ";
  argument = "-i "${input:F1};
  argument = "-o "${output:F2};
  argument = "-o "${output:F3};
  }
DV BM::dv1->BM::P1(
    F1=@{input:"F1.1"},
    F2=@{output:"F2.1"},
    F3=@{output:"F3.1"}
);
DV BM::dv2->BM::P1(
    F1=@{input:"F1.2"},
    F2=@{output:"F2.2"},
    F3=@{output:"F3.2"}
);
…
TR BM::P2(
input F2,
output F4)
  {
  argument = "-T 20 -G 1024 30720 15360 ";
  argument = "-i "${input:F2};
  argument = "-o "${output:F4};
  }
DV BM::dv5->BM::P2(
    F2=@{input:"F2.1"},
    F4=@{output:"F4.1"}
);
DV BM::dv6->BM::P2(
    F2=@{input:"F2.2"},
    F4=@{output:"F4.2"}
);
…
TR BM::P3(
input F2,
input F4,
output F5)
  {
  argument = "-T 20 -G 1024 30720 15360 15360 ";
  argument = "-i "${input:F2};
  argument = "-i "${input:F4};
  argument = "-o "${output:F5};
  }
DV BM::dv9->BM::P3(
    F2=@{input:"F2.1"},
    F4=@{input:"F4.1"},
    F5=@{output:"F5.1"}
);
…
TR BM::P4(
input F5[],
output F6)
  {
```

```
  argument = "-T 30 -G 1024 30720 15360 15360 15360 ";
  argument = "-i "${input:F5};
  argument = "-o "${output:F6};
  }
DV BM::dv13->BM::P4(
    F5=[@{input:"F5.1"},@{input:"F5.2"},@{input:"F5.3"},@{input:"F5.4"}],
    F6=@{output:"F6.1"}
);
TR BM::P5(
input F2,
input F6,
output F7)
  {
  argument = "-T 30 -G 1024 30720 15360 15360 15360 1024 ";
  argument = "-i "${input:F2};
  argument = "-i "${input:F6};
  argument = "-o "${output:F7};
  }
DV BM::dv14->BM::P5(
    F2=@{input:"F2.1"},
    F6=@{input:"F6.1"},
    F7=@{output:"F7.1"}
);
…
TR BM::P6(
input F7,
output F8)
  {
  argument = "-T 50 -G 1024 30720 15360 15360 15360 1024 1024 ";
  argument = "-i "${input:F7};
  argument = "-o "${output:F8};
  }
DV BM::dv18->BM::P6(
    F7=@{input:"F7.1"},
    F8=@{output:"F8.1"}
);
…
TR BM::P7(
input F8[],
output R)
  {
  argument = "-T 30 -G 1024 30720 15360 15360 15360 1024 1024 20480 ";
  argument = "-i "${input:F8};
  argument = "-o "${output:R};
  }
DV BM::dv22->BM::P7(
    F8=[@{input:"F8.1"},@{input:"F8.2"},@{input:"F8.3"},@{input:"F8.4"}],
    R=@{output:"R.1"}
);
```

### Example 4:

```
TR BM::P1(
input F1,
output F2)
  {
  argument = "-T 300 -G 1024 ";
  argument = "-i "${input:F1};
  argument = "-o "${output:F2};
  }
DV BM::dv1->BM::P1(
    F1=@{input:"F1.1"},
    F2=@{output:"F2.1"}
);
…
TR BM::P2(
input F2,
output F3,
output F4)
  {
  argument = "-T 20 -G 1024 30720 15360 ";
  argument = "-i "${input:F2};
  argument = "-o "${output:F3};
  argument = "-o "${output:F4};
  }
DV BM::dv11->BM::P2(
    F2=@{input:"F2.1"},
    F3=@{output:"F3.1"},
    F4=@{output:"F4.1"}
);
…
TR BM::P3(
input F3,
output R1)
  {
  argument = "-T 20 -G 1024 30720 15360 20480 ";
  argument = "-i "${input:F3};
  argument = "-o "${output:R1};
  }
DV BM::dv21->BM::P3(
    F3=@{input:"F3.1"},
    R1=@{output:"R1.1"}
);
...
TR BM::P4(
input F4,
output R2)
  {
  argument = "-T 30 -G 1024 30720 15360 20480 20480 ";
  argument = "-i "${input:F4};
  argument = "-o "${output:R2};
  }
DV BM::dv31->BM::P4(
    F4=@{input:"F4.1"},
    R2=@{output:"R2.1"}
);
...
```