



ALTAIR

ONLY FORWARD

Altair FlowTracer 2024.1.0

Tutorials

Contents

FlowTracer Beginner's Tutorials	3
Create a FlowTracer Project.....	4
Set Command Line Environment.....	4
Create a Project.....	5
Enable a Shell.....	6
Restore the Shell Prompt.....	6
Check Project Information.....	6
Start the GUI Console.....	7
Use the Set Browser.....	8
Add a Job Interactively.....	8
GUI Job Views.....	21
Navigate the Graph.....	25
Command Line Interface.....	29
Flow Description Language.....	34
Remove Older Sets.....	34
The Flow.tcl File.....	35
Build the Flow.....	36
Run the Flow Interactively.....	37
Run a Flow from the Command Line.....	38
Batch Process to Define and Run a Flow.....	39
Create a Complex Flow.....	40
EDA Flows.....	42
Stop the Project.....	44
FlowTracer Advanced Tutorials	45
Create Efficient VOV Scripts.....	46
Write Flows.....	48
The Flow.tcl file.....	48
Build the Flow.....	48
Execute the Flow.....	49
Build a More Complex Flow.....	49
EDA Flows.....	49
Legal Notices	51
Intellectual Property Rights Notice.....	52
Technical Support.....	56
Index	58

This chapter covers the following:

- [Create a FlowTracer Project](#) (p. 4)
- [Flow Description Language](#) (p. 34)
- [Stop the Project](#) (p. 44)

Create a FlowTracer Project

The purpose of this tutorial is to guide one through the creation and use of a small FlowTracer project using a combination of the Command Line Interface (CLI) and the Graphical User Interface (GUI).

Preliminaries

- Read the overview section to become familiar with some of the FlowTracer terminology: files, jobs, nodes, sets, etc.
- Install FlowTracer (if it hasn't already been done).

When you're ready, start the tutorial in the next section.

Goals

At the end of the tutorial, you will know how to fire up a FlowTracer project and to register a set of jobs into FlowTracer to create a dependency graph that you can view in the console.

The intention is to become comfortable with creating projects, adding nodes to the dependency graph by registering programs and files into FlowTracer, viewing the dependency graph in the console, and running the project. And finally, stopping the project and clearing out the dependency graph and the project.

Tasks in This Tutorial

Set Command Line Environment

You need to have your shell command line environment set properly in order to use FlowTracer.

This includes changing your PATH environment variable so you can run the installed executables, and adding two environment variables that are used by the Altair Accelerator programs.

You can set your command line environment by sourcing a setup file created by the installation. You will source the setup file that matches your platform and shell.

Assuming that FlowTracer is installed at the path `\opt\altair\vov\1212.10`, this table shows the way to source the setup file so that your shell environment is correct.

Platform Type	Shell	Command to Source File
UNIX	csh tcsh	<code>source /opt/altair/vov/1212.10 platform/etc/vovrc.csh</code>
	bash sh ksh zsh	<code>source /opt/altair/vov/1212.10 platform/etc/vovrc.sh</code>
Windows	DOS	<code>\opt\altair\vov\1212.10\win64\bat \vovinit.bat</code>

Create a Project

FlowTracer keeps track of the files and jobs that make up the flow you want managed. The collection of such files and jobs constitutes a "project". Each project has one dependency graph that encodes the runtime trace of the jobs and files in the project. Each dependency graph of a runtime trace has one running server process that manages it.

You begin using FlowTracer by creating a project and starting its server. If a project was already created but not running, then starting its server would be all that was needed.

To create a project, you must choose at least:

- a name for the project; any alphanumeric string can be used
- a host to run the server

Optionally, you may also specify

- a directory that will hold the "server working directory" (.swd) for the project. This .swd directory will contain the system control files used by FlowTracer. The default location that holds the FlowTracer server working directory is inside the vov directory in your home directory (~ /vov) on UNIX, and in \$VOVDIR/local/swd on Windows.

For this tutorial, we will use the default location for holding the .swd directory.

1. To create and start a project, use `vovproject create` as shown here:

```
denby1 (no project) DEFAULT+P4+P4 ~/Perforce > cd ..
denby1 (no project) DEFAULT+P4+P4 ~ > vovproject create tutorial_denby
Creating a new project:
  Directory  /home/denby/vov
  Type      generic
  Product   auto
  Name      tutorial_denby
  Port      any
  Web port   0
  Guest port 0
vovproject 11/22/2019 05:45:21: message: Creating server directory "/home/denby/
vov/tutorial_denby.swd/."
vovproject 11/22/2019 05:45:21: message: Created setup file '/home/denby/vov/
tutorial_denby.swd/setup.tcl'
vovproject 11/22/2019 05:45:21: message: Copy all files from /remote/release/
VOV/2019.01_71758_Apr25/linux64/etc/ProjectTypes/generic
vovproject 11/22/2019 05:45:21: message: Updating autostart/
start_vovnginxd.tcl...
vovproject 11/22/2019 05:45:21: message: Warning: the path permissions
for taskers are 040777 instead of 0777
vovproject 11/22/2019 05:45:21: message: Starting a VOV server
for project tutorial_denby@denby1
PORT=any,WEB_PORT=0,READONLY_PORT=0
```

This command creates all necessary project control files in the server working directory and starts the server process for the project.

2. You can use the list option of the `vovproject` command to see what projects are available and see their status. At this point, you will see that the tutorial project is running.

```
% vovproject list
```

Enable a Shell

To be able to work with the newly created project and the running server, you need to enable your shell with:

```
denby1 (no project) DEFAULT+P4+P4 ~ > vovproject enable tutorial_denby
vovproject 11/22/2019 05:48:25: message: Enabling project 'tutorial_denby'...
```

This command essentially sets two important environment variables:

- VOV_HOST_NAME
- VOV_PROJECT_NAME

Restore the Shell Prompt

There is another change that happens when you enable a project. Running `vovproject enable` changes your shell prompt.

The new prompt contains the name of the local host, name and host of the current project, the current environment, and the last two components of the current directory.

For example:

```
[orange]% vep
orange tutorial@apple BASE john/test > _
```

The effect of this command is purely cosmetic; its purpose is to make you aware of the current environment and of the current project. Since it modifies the current shell, it is implemented as an alias for `csh/tcsh` users, and as a shell function for `sh/ksh/bash`.

To restore your original prompt, use the command `veprestore`:

```
orange tutorial@apple BASE john/test > veprestore
[orange]% _
```

Check Project Information

For basic information about the status of the server, use either the command `vovproject info` (or the shorter equivalent `vsi`).

```
[denby@denby1 ~]$ vsi
Vov Server Information - 11/22/2019 05:50:19
tutorial_denby@denby1:10813      | URL: http://denby1:10813
-----
Jobs:                            0 | Workload:
Files:                            0 | - running:                0
Sets:                             15 | - queued:                 0
Retraces:                          0 | - done:                   0
                                   | - failed:                 0
-----
Taskers:                           1 | Buckets:                   0
- ready:                            1 | Duration:                  0s
Slots:                               8 | SchedulerTime:            0.00s
```

```
-----  
TotalResources:          10 | Pid:                    59021  
                          | Saved:                 4m58s ago  
                          | Size:                 27.00MB  
-----  
                          | TimeTolerance:        1s  
-----
```

For now, do not be concerned about the information returned by this command; it is being used here to check that the server process was started correctly.

Start the GUI Console

Now that you have created the project and started the server, you can begin to use FlowTracer. For this tutorial, you will use the GUI console, and commands from the shell. Be aware that the console functionality can also be accessed from a browser using the flow management web application. The GUI is visually oriented compared to the browser interaction which is list and text oriented.

Start the graphical user interface (or GUI) with the command `vovconsole`.

```
% vovconsole &
```

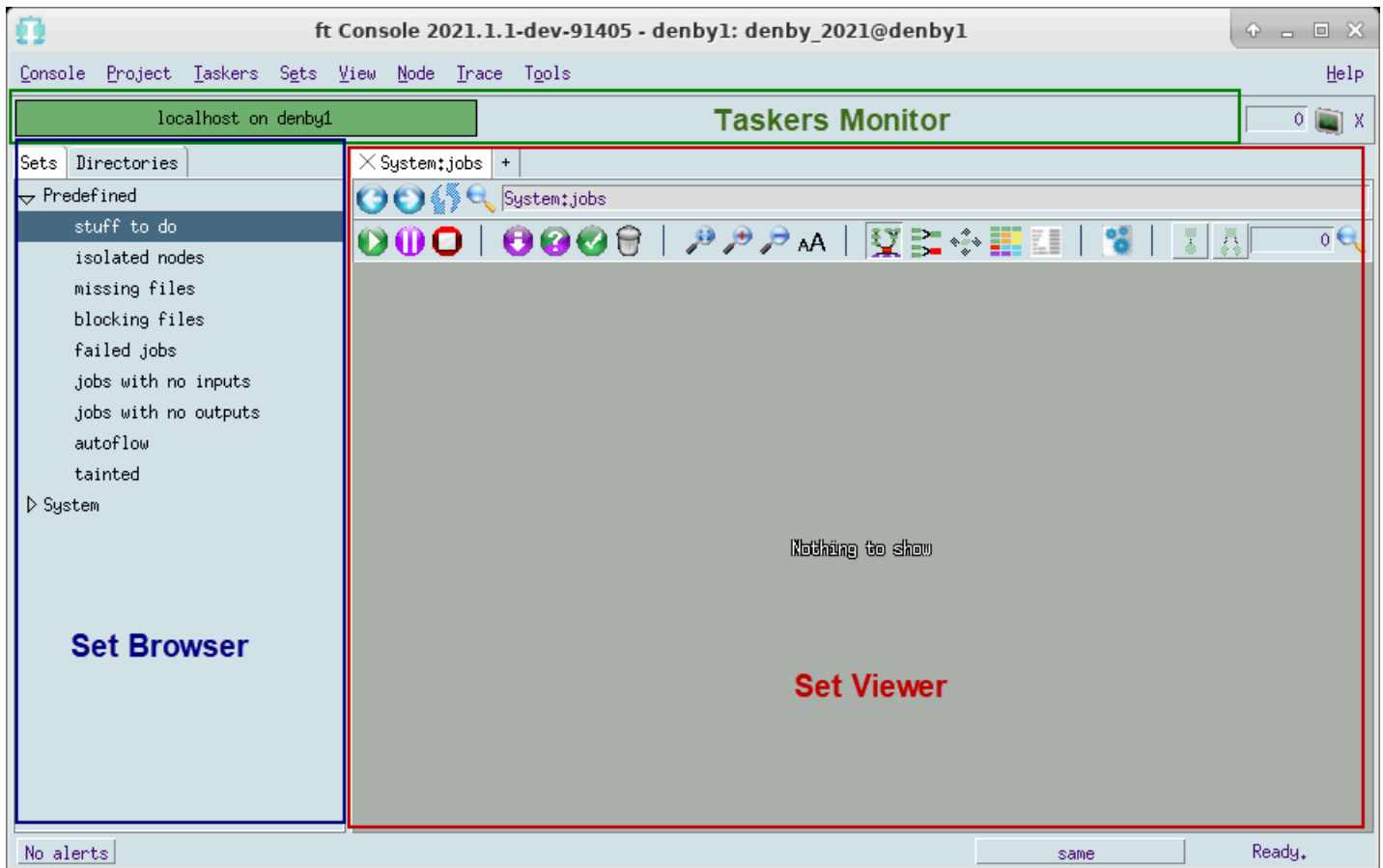


Figure 1: Initial console screen

The program "vovconsole" stays running while the GUI console is active so it is best to run it in the background to let the command line be available for more commands.

At this point, as no files or jobs are yet registered, the console will show a grey background with an empty Set Viewer on the right side, and two top-level elements in the navigation list in the Set Browser panel on the left.

Use the Set Browser

In this section you will learn how to navigate around sets using the set browsing control. Later, when there are interesting sets defined, the Set Browser will be used to choose which set of nodes to view in the Set Viewer.

1. Click on the **Sets** tab.
2. Open the **Predefined** folder in the navigation control by clicking on the right-arrow control icon.
3. Double-click on the set **stuff to do** to display it.

At this point, you should be seeing the string "Predefined:stuff to do" as the tab label above the Set Viewer. This set is empty, so no nodes appear in the Set Viewer. With no jobs or files registered with FlowTracer yet, viewing of different sets is not interesting - they are all empty sets. The important point is to notice the names of the System and Predefined sets, and how to navigate to them.

4. Expand the **System** folder in the Set Browser.
5. Double-click on the set **nodes** to display it.

Leave this set on display. Later, when jobs are registered with FlowTracer, this display will show added nodes that represent the added jobs.



Note: The display will show changed states of nodes in the display but it will not change the group of nodes on display unless you click the **Refresh** icon.

Add a Job Interactively

In this tutorial, we will build a simple flow graph one job at a time, interactively from the command line. This is to demonstrate the basic building blocks for adding jobs to a flow graph, and how we can monitor a flow using the FlowTracer GUI program "vovconsole". This is not to demonstrate production techniques for building a flow.

You will register jobs interactively with FlowTracer to have FlowTracer build its flow graph one job at a time. This will make it easy to see what is going on. In a production situation, jobs are not registered into the flow graph interactively. Instead, the jobs are registered into the flow graph by way of batch scripts or by processing control files. The batch style of registering jobs will be shown later in the tutorial.

Objective

- Verify that FlowTracer is properly setup.
- Become familiar with named environments.
- Learn more about the concept of runtime tracing.
- Become comfortable using the FlowTracer GUI for managing the jobs.

Use the "cp" Program to Emulate a Tool

You will use the UNIX copy program `cp` to emulate a more useful tool having an input and output. The `cp` command comes with UNIX, and FlowTracer supplies a script file called `cp.bat` which allows you to also run this tutorial on Windows.

The UNIX program `cp` reads an input file and copies it to an output file. It can be used for a very simple job to create a backup of a file.

```
cp input-file output-file
```

Consider a job that is a typical computer task using a tool named TRANSFORM:

```
Job 1: TRANSFORM source-file expanded-file
```

This job is a generic one that reflects what most tools do. It reads an input file and generates an output file based on it.

We will use the `cp` program during this tutorial as a fast and low cost tool to demonstrate how to register jobs into the flow graph, and how to monitor and control the work of FlowTracer, the flow manager.


Here is the above TRANSFORM job emulated using the UNIX `cp` command.

```
Job 1: cp input-file output-file
```

The goal of the job is to create the output file. The output file depends on the tool to generate it from the input file. If the input file changes, then the output file is out of date, and is INVALID using flow terminology. When the input file is changed, then the job's goal to create the output file is triggered. To reach the goal, the tool must run, process the input, and create a version of the output that is up to date. This makes the output file VALID.

Create a Project Directory

The project directory is the area where the data files for your project are stored. When creating this directory, place it on a file system which is available on the network. Somewhere in your home directory is usually a good starting point.

 **Note:** Do not create the data directory inside the FlowTracer software installation, even if you have installed the software under your home directory. By default, the files under `$VOVDIR` are excluded from the graph, and your jobs will fail because they appear to have no outputs.

To create the project directory, execute the following:

```
% cd  
% mkdir simple_test  
% cd simple_test
```

Register One Job from the Command Line

To register a job with FlowTracer so that the inputs and outputs will be dynamically discovered as the job runs (known as runtime tracing), it is necessary to define the job by way of a wrapper program. In this section of the tutorial, we are registering jobs with FlowTracer interactively, from the command line.

At the command line, the program `vw` is the wrapper to register a job. `vw` is a program that takes a parameter that is the command line defining the job.

Here is the logical way of thinking about running the `vw` wrapper.

```
Usage: vw <command line that runs a job>
```

Next is an example of using `vw` to register a job to compile a C program. The job consists of running the clang tool to compile a source file into an object file.

```
% vw clang myprogram.c
```

By using the `vw` wrapper, runtime tracing of the job is established as it is added to the flow. runtime tracing is the feature of FlowTracer where it discovers and notices the resulting output file `myprogram.o` without you having to mention it in the command line, and without you having to explicitly tell FlowTracer about it. It can do this because the job is run within a wrapper that checks for implicit inputs and outputs used at runtime, and tells FlowTracer about them.

For this tutorial, you will be using `cp` as our emulation tool and using a file named "aa" as the primary input file. You must create a primary input file for our emulation. Do this command to create an empty file "aa" which will be our primary input file.

1. You must create a primary input file for the emulation. Do this command to create an empty file "aa" which will be your primary input file.

```
% touch aa
```

You will register a job that emulates transforming an input file to an output file by using the `cp` command. This is the job command that will be registered.

```
cp aa bb
```

2. Do the command below to register this job, by calling the `vw` wrapper program and passing it the command of the job.

```
% vw cp aa bb
```

This registers the job with FlowTracer. FlowTracer then runs the job. When it executes, the wrapper sends messages to the FlowTracer server describing the inputs and outputs of the program `cp`. As the job runs, you will see activity in the Set Viewer. Make sure to be looking at the **System > nodes** set.

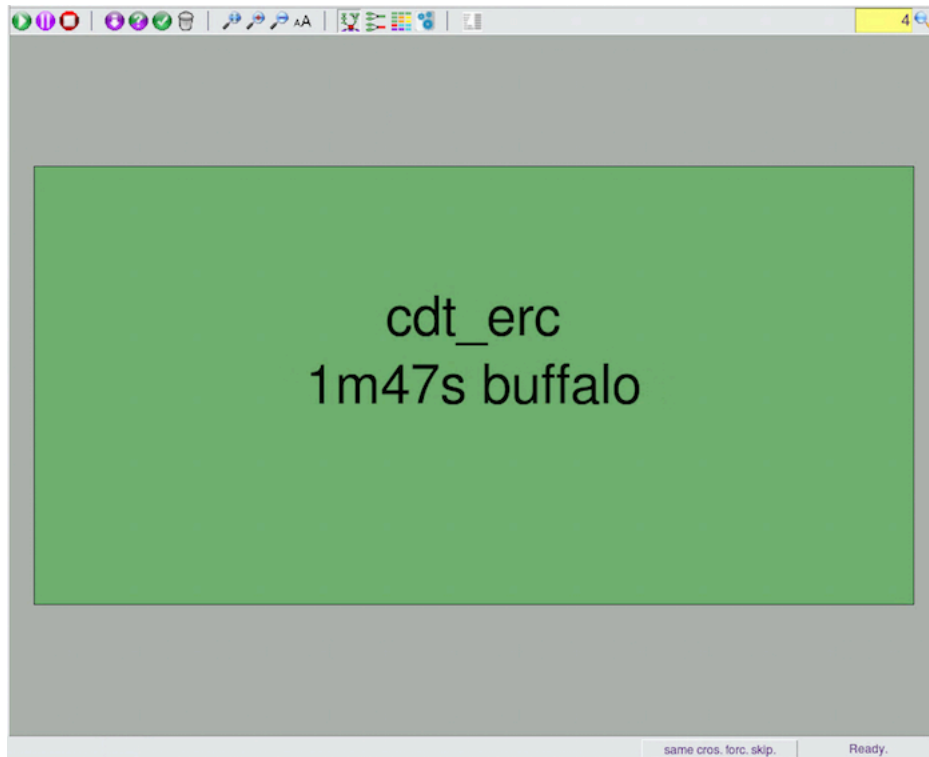


Figure 2:

You will see a graph such as the one above. This shows the job node as a single green rectangle. You can see the number "4" in the upper right of the Set Viewer. This is reporting on the number of nodes in the set on display. The file nodes are not on display.

3. To turn on the display of file nodes, right click in the background of the Set Viewer and click **Show/Hide** to open a submenu where you can toggle on the **Show Files** option.

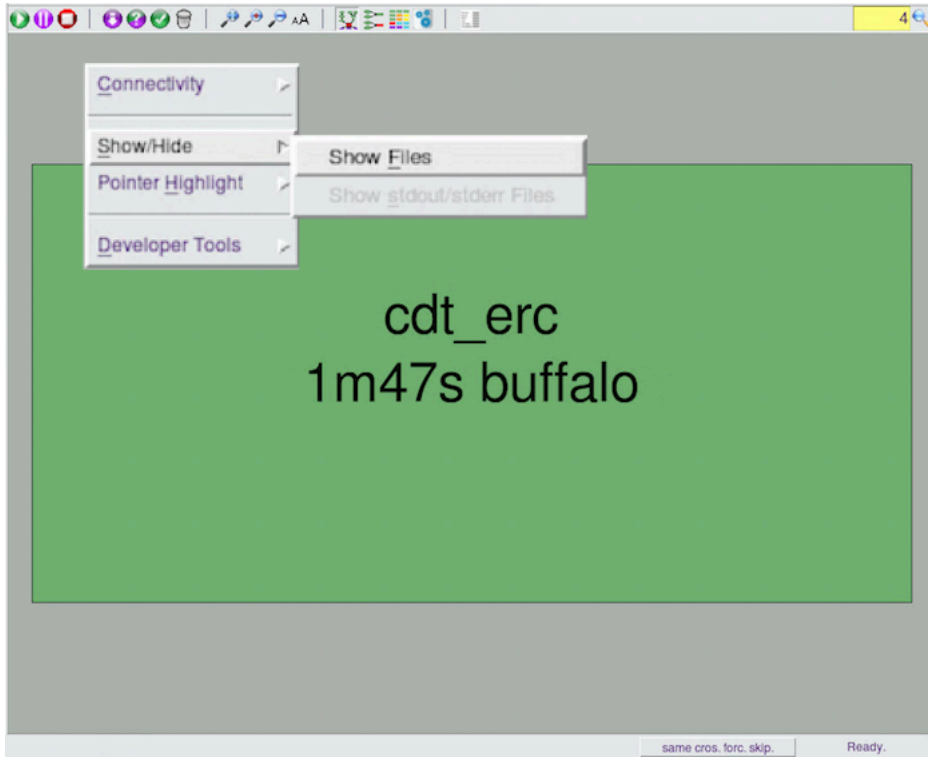


Figure 3:

After turning on the display of file nodes, you will see the four nodes in the flow graph. This represents the dependency graph that is now encoded into FlowTracer's flow.

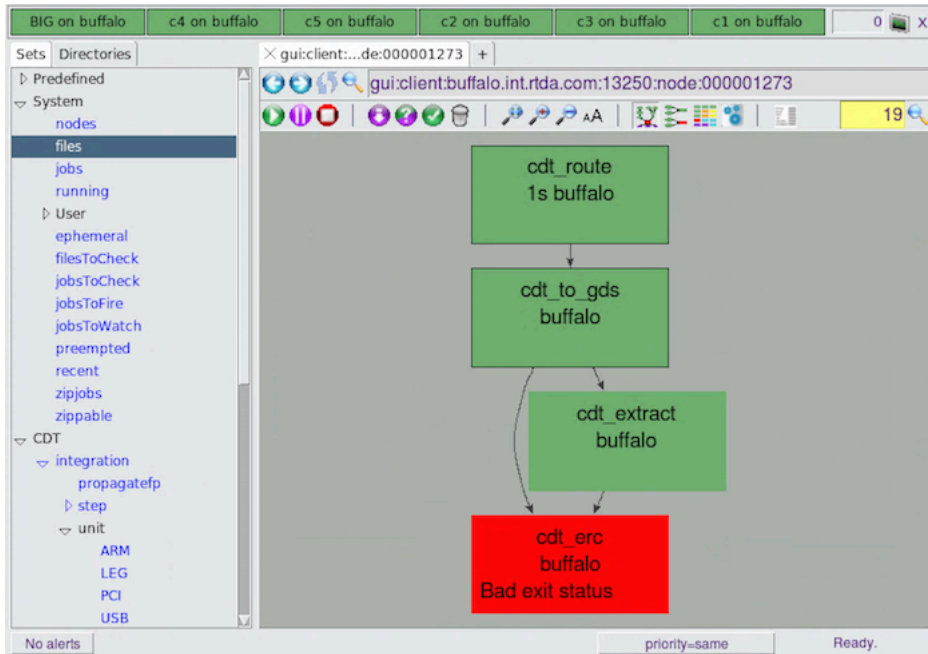




Figure 4:

 **Note:** If your display is not showing up completely within the Set Viewer pane, you can fit the graph to the available area by clicking on  to fit the view.

4. Repeat the registering of the `cp` command if you missed seeing the changes to the display as it ran.

```
% vw cp aa bb  
% vw cp aa bb
```

This appears to register the job again. But because the command is the same string, and we are in the same directory, it is recognized as the same job as an existing job, and a new job is not created.

The circles represent files, the rectangle represents a job, in this case the job of copying file `aa` to file `bb`. The arcs represent the input/output relationships between files and jobs.

The green color means that the files and the jobs are up-to-date.

In this case, there is one input file `"aa"` and one output file `"bb"`. Depending on your setup, you may get additional inputs, such as the file `"cp"`.

This `"cp"` file exists as an input in the dependency graph because the program `cp` is a dependent element of the task. If the version of the program changes, then the result of the task could be different. This was noted by FlowTracer even though we did not explicitly tell FlowTracer about this dependency. This is an example of FlowTracer performing runtime tracing to figure out all the dependencies, even if you do not register them fully.

In production you can exclude program files from the graph. For this tutorial, you will see that `"cp"` input file as a dependency node. All the circles and the rectangle should be green at this point, meaning that they are all up-to-date, or, `"VALID"`.

Add More Jobs to the Flow

A project is normally made up of many jobs that work together toward the goal of the project. A given job may depend on the output of another job, and in turn may create output that is needed by a downstream job. Each job must be run in the proper sequence in order to reach the project's goal.

Next you will add more jobs to this project to emulate that aspect of dependency. Continue to use the `cp` program to emulate all the various tools used in your jobs.

Consider a project having these logical job steps using four different tools:

```
Job 1: TRANSFORM source-file expanded-file  
Job 2: TRANSLATE expanded-file translated-file  
Job 3: SORT translated-file sorted-file  
Job 4: ARCHIVE translated-file archived-file
```

This reflects a project goal of creating two final result files that are formed by running processing tools in the proper sequence, based on a single input file.

Here it is again, using simple file names:

```
Job 1: TRANSFORM aa bb  
Job 2: TRANSLATE bb cc  
Job 3: SORT cc dd1
```

```
Job 4: ARCHIVE cc dd2
```

1. Emulate this project using `cp` with this job list:

```
Job 1: cp aa bb  
Job 2: cp bb cc  
Job 3: cp cc d1  
Job 4: cp cc d2
```

This project has exactly the same dependency graph as the one above. We have emulated a complex project with this technique of using `cp` with simple, empty files.

2. Add the extra jobs to the flow managed by FlowTracer to register this larger project. Run the `vw` wrapper program with the job command parameters that define the additional jobs. Execute these commands:

```
% vw cp bb cc  
% vw cp cc dd1  
% vw cp cc dd2
```

This creates a flow that is getting more complex and has more dependencies for FlowTracer to manage. If file "aa" is changed then files "bb", "cc", "d1" and "d2" all become INVALID.

FlowTracer will notice if that happens and mark the files as INVALID. FlowTracer can also schedule and dispatch the jobs to run in the proper sequence to make the INVALID files VALID.

At this point the graph should look similar to the one in shown below. Minor differences in the horizontal position of the nodes are to be expected.

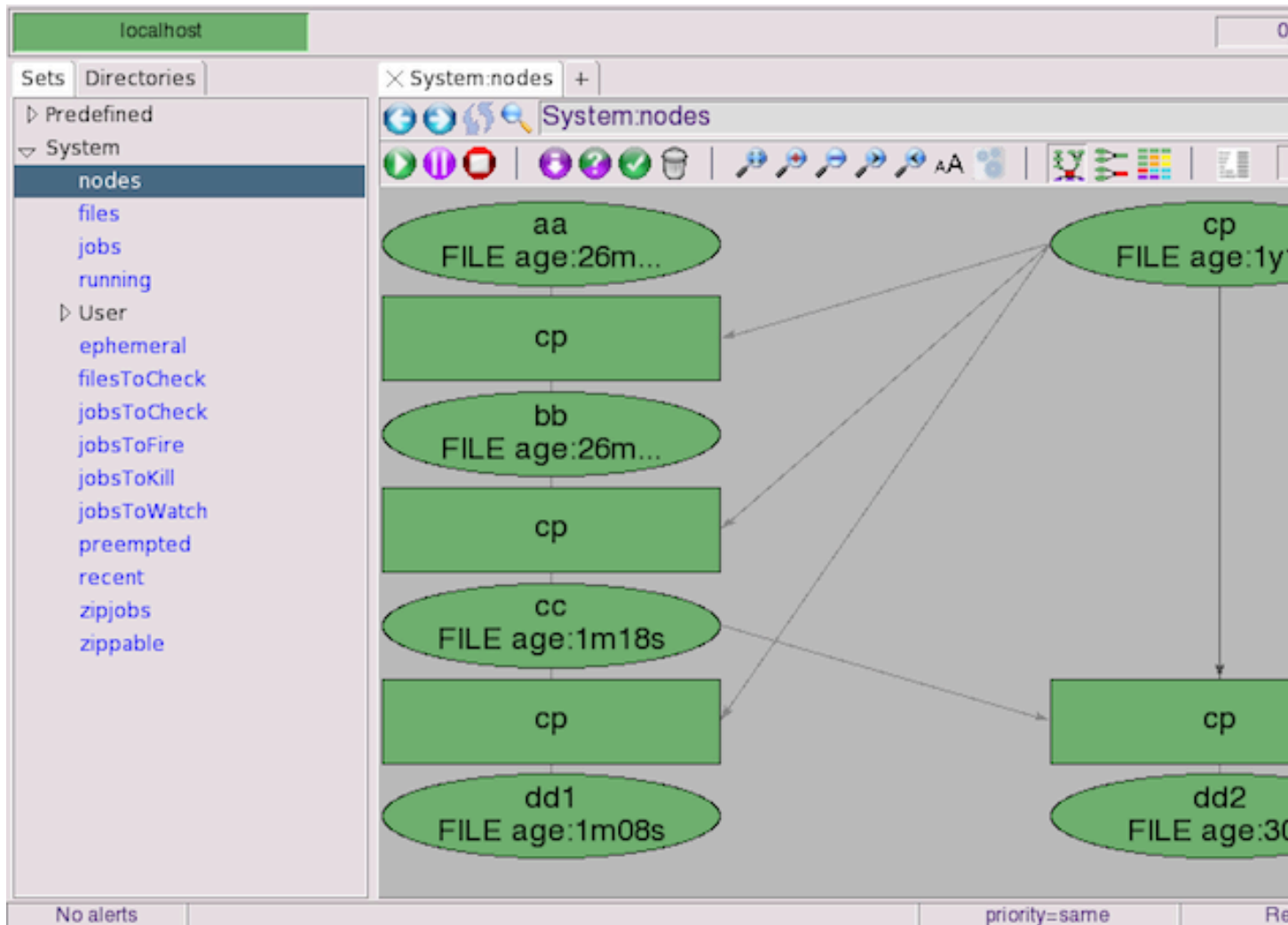


Figure 5:

What you have done in this short exercise is to register jobs into the flow for FlowTracer to manage. You did this by running the wrapper program `vw` and giving it a command line that defines the job. The jobs use the program `cp` as an emulation of a program that processes an input to create an output.

This demonstrates an interactive way to register jobs, but is not promoting this as the way to register jobs in a production environment. The intent is to demonstrate what the display of the flow graph looks like as a job is added to the flow. You have now seen how the data structure within FlowTracer holds the dependency graph between programs and files, how FlowTracer reports on the state of files using shapes and colors. You have seen how the FlowTracer GUI helps you visualize the state of the flow graph.

Change Dependent Input File

We have built up a dependency graph for a task that requires running four jobs in a sequence to produce two result files ("`dd1`" and "`dd2`") based on a starting file ("`aa`").

This establishes a model that FlowTracer will use to schedule and deploy the jobs as the dependent input files change.

1. While looking at the GUI console, touch the file "bb" to give it a timestamp of now. Touching the file causes its timestamp to be later than the timestamps on files "dd1" and "dd2". This emulates changing file "bb". The two output files "dd1" and "dd2" are now out of date relative to file "bb".
You will see the Set View display change. The out of date output file nodes will change color from green to purple. The dependent jobs of copying file cc are also out of date and change color. The nodes in purple are INVALID.

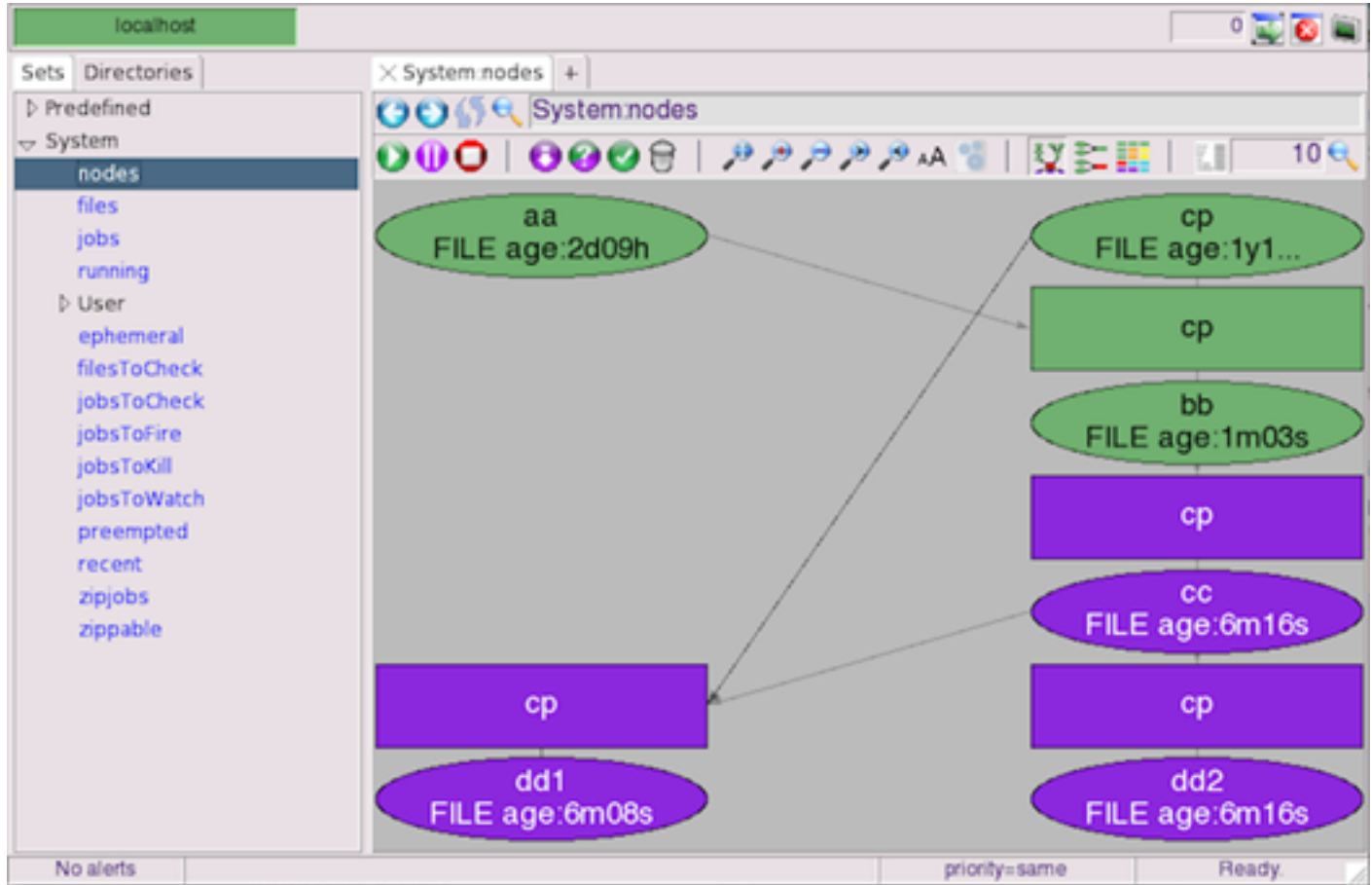



Figure 6:

FlowTracer has noticed the change of file "bb" (the timestamp is recent) and is aware of what nodes are INVALID and knows what needs to happen to make them VALID.

2. Click **Run**  in the action bar at the top of the Set Viewer panel to request that FlowTracer brings the nodes up to date. This will cause the "cp" programs to run. While they are running, their nodes in the display will turn yellow. When the jobs and files become up to date, they turn back to green to indicate they are VALID.
3. Repeat this sequence, again, but touch file "aa" instead of "bb". The display shows the dependent nodes as INVALID (purple).
4. Click **Run** again. FlowTracer dispatches the jobs in sequence to bring all the nodes up to date.

Remove Dependent Input File

1. Delete the file "aa" and notice the display change.
The "aa" node changes to a brown color to indicate that the file is missing.

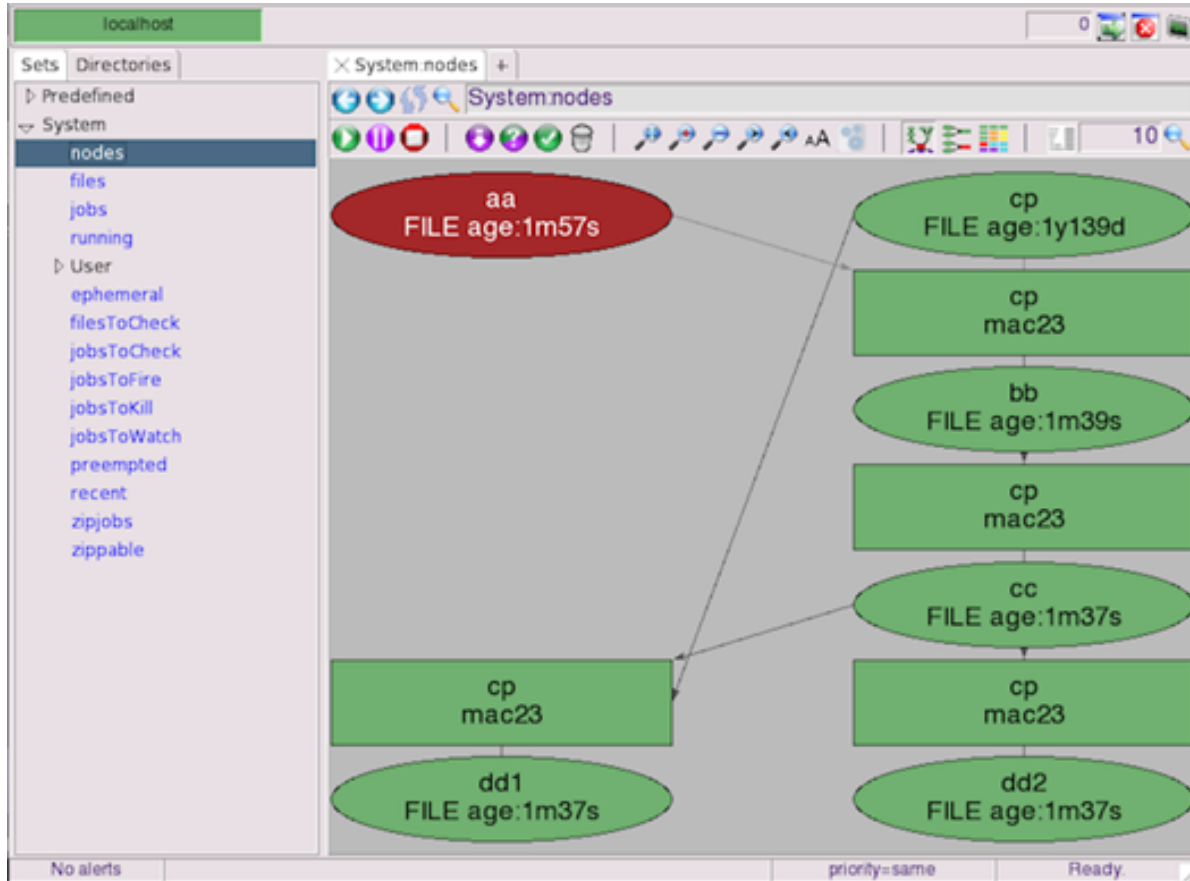


Figure 7:

You can see the FlowTracer has changed the status of the nodes to indicate the state it has noticed - the dependent input file does not exist.

The dependency graph does not show that dependent files are out of date (INVALID) when an input file is missing. This is the proper response to a missing input file. The input file is not changed and dependent jobs do not need to be run to produce new output files.

2. Touch file "aa" to put it back into existence.
This causes the file "aa" to become changed (timestamp is more recent). Notice that the display changes again. This time the node for file "aa" changes to a slightly different hue of green and the dependent nodes turn purple.
The different colored green indicates that the file was recently changed. This subtle state is shown with a subtle color variation.

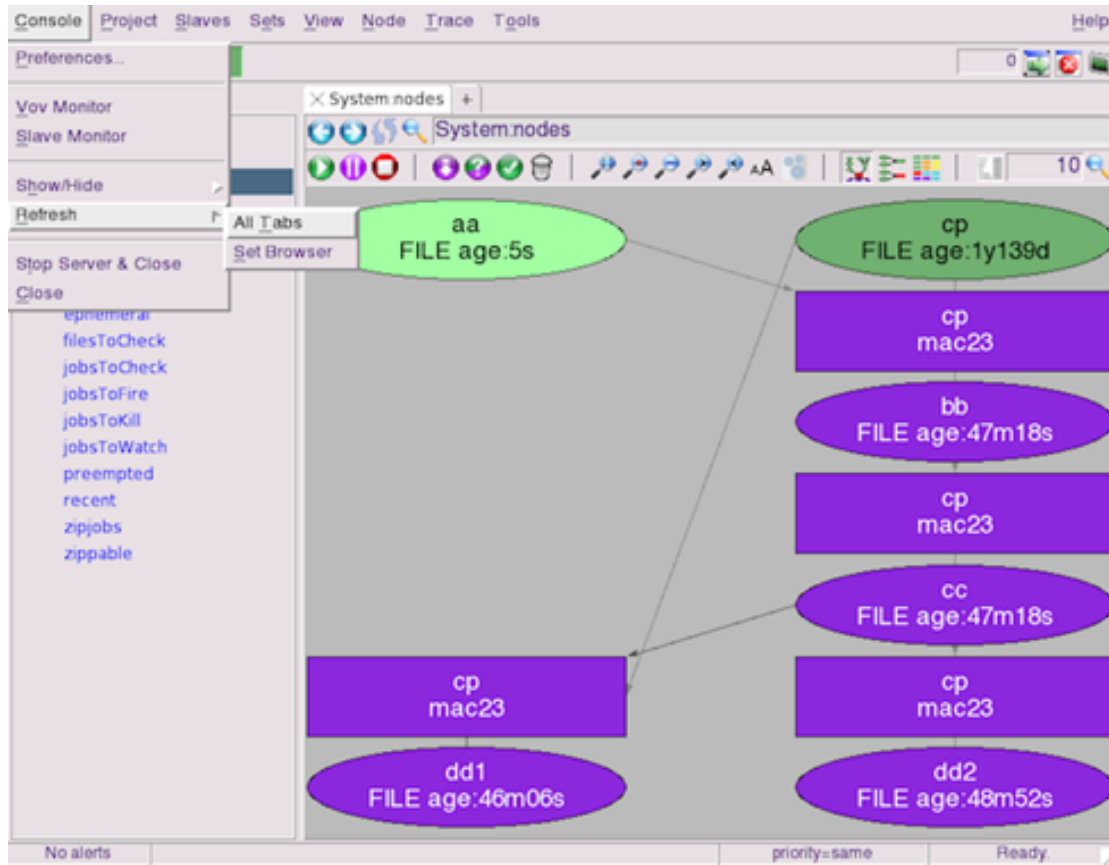


Figure 9:

4. Click **Run** to see the dependent jobs get dispatched, which causes the dependent files to be created again and become VALID.

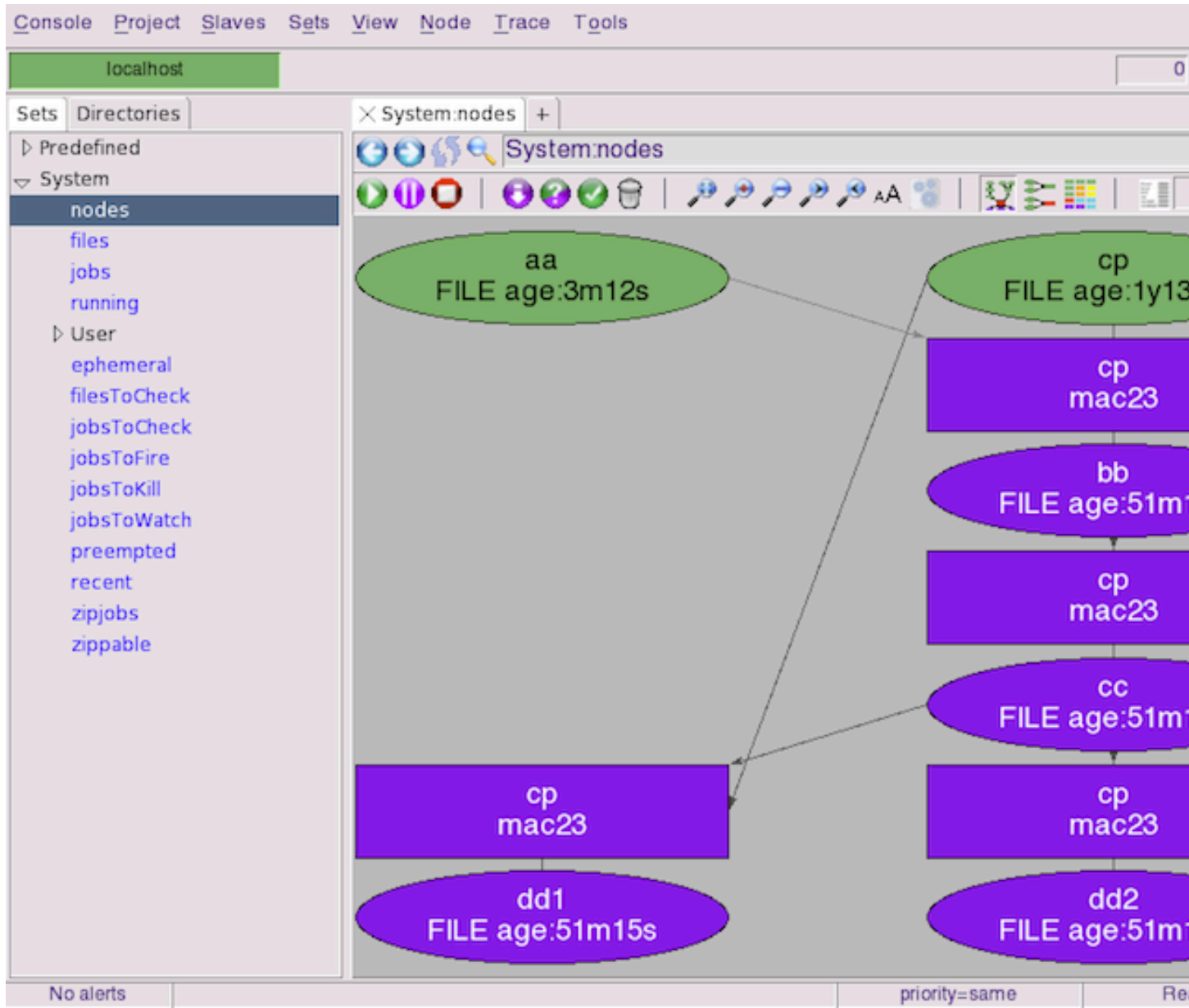
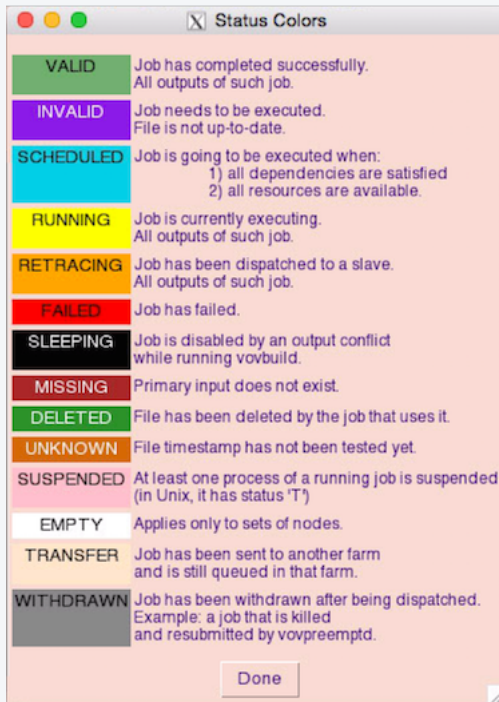


Figure 10:

 **Tip:** You can see a table showing all the node colors and their meanings by clicking on the top menu **Help > Status Colors**.



Color	Description
VALID	Job has completed successfully. All outputs of such job.
INVALID	Job needs to be executed. File is not up-to-date.
SCHEDULED	Job is going to be executed when: 1) all dependencies are satisfied 2) all resources are available.
RUNNING	Job is currently executing. All outputs of such job.
RETRACING	Job has been dispatched to a slave. All outputs of such job.
FAILED	Job has failed.
SLEEPING	Job is disabled by an output conflict while running vovbuild.
MISSING	Primary input does not exist.
DELETED	File has been deleted by the job that uses it.
UNKNOWN	File timestamp has not been tested yet.
SUSPENDED	At least one process of a running job is suspended (in Unix, it has status 'T')
EMPTY	Applies only to sets of nodes.
TRANSFER	Job has been sent to another farm and is still queued in that farm.
WITHDRAWN	Job has been withdrawn after being dispatched. Example: a job that is killed and resubmitted by vovpreemptd.

Figure 11:

This demonstrates the way in which FlowTracer manages the dependency graph in order to schedule and deploy jobs as needed to run the trace so that all nodes become VALID. Using the GUI console, you can watch the display change to view the data structure that FlowTracer manages and to watch progress as FlowTracer dispatches dependent jobs and dependent files are updated.

GUI Job Views

Now that you have a small flow, you can familiarize yourself with the console and its various views.

In any view that you choose, the following features are available:

- Hover the mouse over a node to display a descriptive label.
- Right click on a node to get a node operations menu.
- You can select multiple nodes with a rubber-band action: point over blank space, left-click and drag, release: all nodes completely contained in the rectangle will be highlighted. Now, all operations in the pop-up menu apply to all selected nodes.
- Double click on a node to open the Node Editor, which displays the properties of the node.

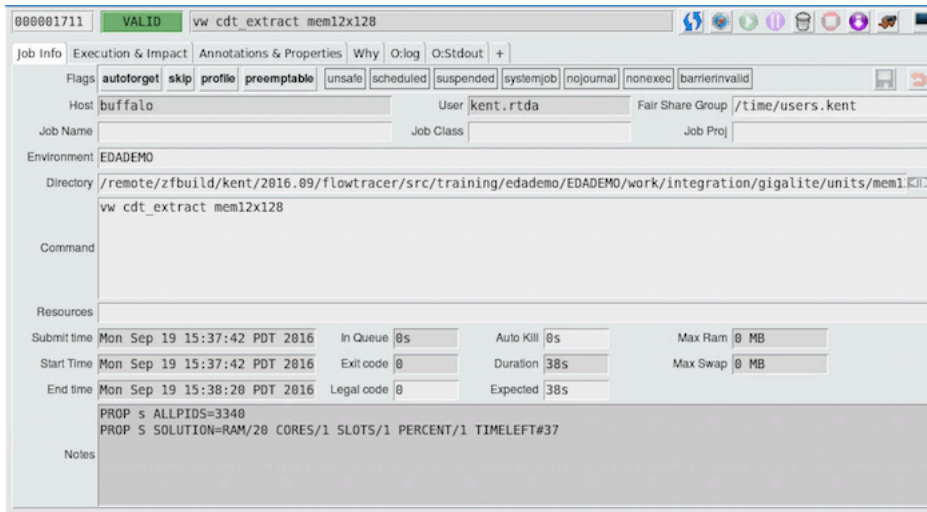



Figure 12:

- While the Node Editor is still open, select other nodes by clicking on them. You will see the information in the Node Editor change as you select different nodes.

Vertical Graph View

Open the Vertical Graph view by clicking the  or by clicking the letter **G**.

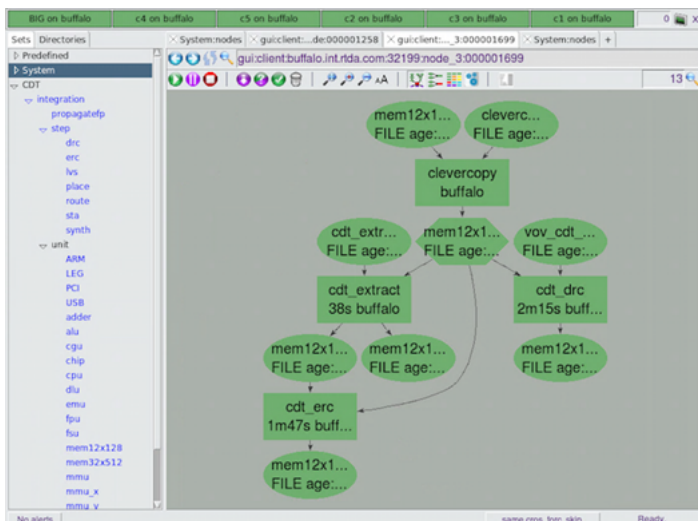


Figure 13:

Horizontal Graph View

Open the Horizontal Graph view by clicking the  or by clicking the letter **H**.

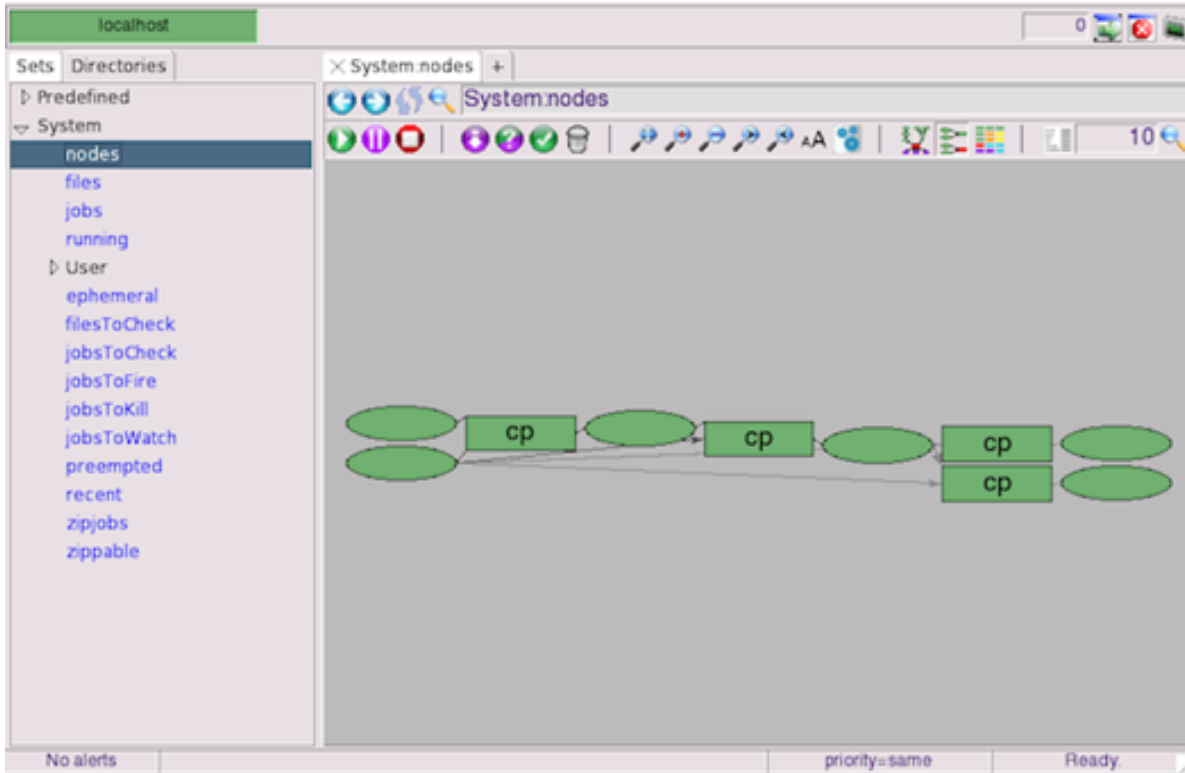


Figure 14:

Grid View



Open the Grid view by clicking  or by clicking the letter **Q**. This is an alternative graphical representation for the dependency graph in which arcs are not shown and the nodes are compactly arranged in a non overlapping grid. This view is effective when you have hundreds or thousands of nodes to show.



Figure 15:

Stat View

Open the Stat view by clicking  or by clicking the letter S.

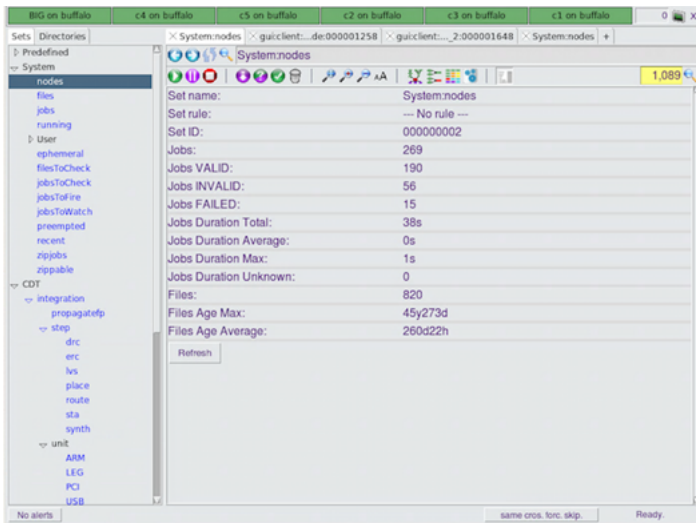



Figure 16:

View Graph Subsets

The previous views have only shown the complete dependency graph, that is the set "nodes". It is valuable to look also at smaller subsets of the graph.

1. Select the **Graph** view.
2. In the Set Browser, in the System folder, double-click on the sets **jobs** and **files**.
3. Under the Predefined directory are many sets that will be useful when dealing with real life projects. The primary sets are **Stuff to do** and **Failed job**.
These sets are currently empty. Later in the tutorial you will see how they can be used.
4. Go back to the set **System:nodes**.
5. Right click the node representing the file **bb** to show the pop-up menu.
6. Select **Connectivity > Selection Alone**.
7. Expand the graph around this node with the pop-up menu **Connectivity > Expand Selection**.

 **Tip:** Many common commands are bound to keyboard accelerators. For example, the operation you just performed (showing a node alone, then expanding the graph) can also be performed by typing a while the mouse is over the node to view the node alone and by typing **x** to expand the node.

8. Display only the node **bb**. It may be useful to look at the inputs of node **bb** and their inputs, and their inputs and so on. The set of all transitive inputs of a node is called the "up-cone" of the node. The accelerator for up-cone is **Ctrl-u**.
9. Repeat the previous exercise but get the "down-cone" this time, that is the set of all outputs of a node, and their outputs, and so on. The accelerator for down-cone is **Ctrl-d**.
As you view selected subsets of the dependency graph, FlowTracer creates new sets. These are visible if you refresh the Set Browser by clicking **Set > Refresh browser**.

Navigate the Graph

Make Changes to a File and Run it

Edit, modify and save **aa** and watch what happens to the dependency graph.

All the nodes dependent on **aa** change color as they are no longer up to date with respect to **aa**. The purple color indicates that the nodes are invalid with respect to their inputs.

Run the Jobs

You have built the graph by executing the "tools" (emulated by **cp**) interactively under the control of the **vw** wrapper. By using that wrapper, you have established runtime tracing. With runtime tracing turned on, FlowTracer discovers the inputs and outputs at runtime as the program is run. It builds a dependency graph of the files related to the program. The flow holds a data set that defines the dependency graph, the jobs, and the current state of files. FlowTracer is now ready to execute the jobs in the flow, based on the dependency graph and the state of files in the system. It can schedule and deploy jobs that need to be run because they use an input file that has changed. This process is called "retracing".

1. Go to the graph view.
2. Point at the node **bb**.
3. Right-click and hold to display the menu and select **Run**.

A request to the FlowTracer server to bring the file bb up to date. This means re-executing the job vw cp aa bb.

To accomplish this task, the server selects the fastest tasker in the network that can execute the job. When processing a run request, FlowTracer takes advantage of potential parallelism by sending multiple independent jobs to the available taskers. Depending on the number of taskers connected to your project, you may or may not see parallelism in action. This illustrates the process of bringing a particular file up-to-date. More commonly, you will want to bring the entire design up to date, or the entire set of nodes you are looking at.

4. Select the set you are interested in; for example, choose the set **"nodes"** in the Systems folder.
5. Select the **Graph** view in the Set Viewer.
6. Click on the **retrace** icon to bring the current set up to date. If all elements in the set are already VALID (green) nothing needs to be done.

 **Note:** You can also run individual nodes using the pop-up menu, or the keyboard shortcut "r".

Navigate the Graph

Navigating the graph means moving from a node to another following the input/output dependencies. You can navigate the graph using the **Navigation** dialog:

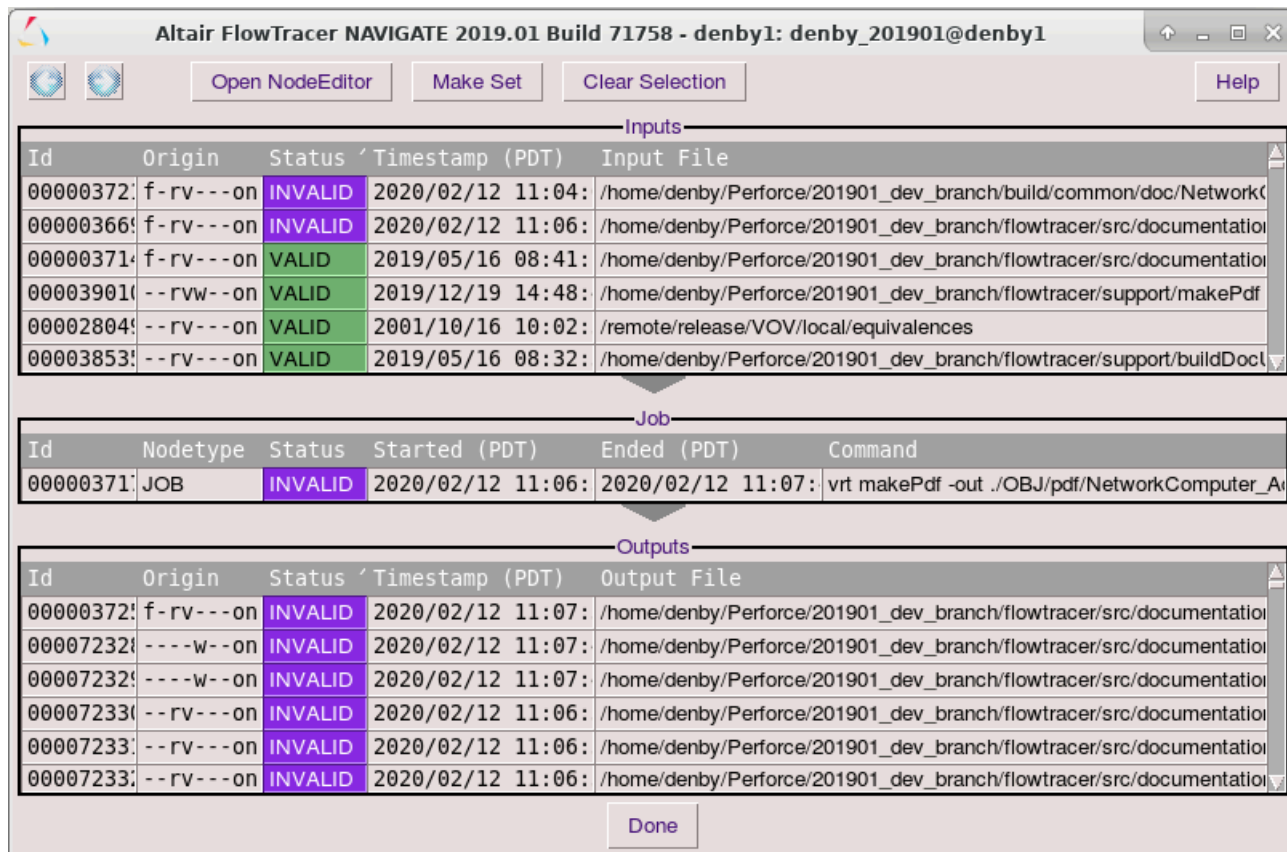



Figure 17:

Invoke the graph by right-clicking on a node and selecting **Connectivity > Navigate** or by clicking .


The dialog is divided into 3 parts:

1. the top shows the inputs
2. the center shows the current node
3. the bottom shows the outputs

You can select any of the rows in the **Navigation** dialog. When one or more rows are selected, the buttons at the top get enabled. Using these buttons, you can either edit a selected node, create a set containing all the selected node, or clear the selections.

Further, right clicking on any node will popup a context menu which can be used for various operations on that node.

Multiple rows can be selected by holding the mouse left button down and dragging the mouse across the rows you want to select.

 **Note:** The rows in any of the sections in the **Navigation** dialog can be sorted by any column by clicking on the column header.

Determine Reason for Invalid Node Status

1. Edit file aa, save the changes and wait for the graph to turn purple.
2. Double-click on cc and in the **Node Editor** window select the "**Why?**" tab.
You will see the reason why FlowTracer thinks that the file cc is not up to date because the transitive input aa has been changed.

 **Tip:** You can get the same information from the command line with the command vsy.

Analyze Impact

You can analyze the consequences of changing a file with the Impact analysis function.

1. Click on a node to select it, for example, aa.
2. From the menu, click **Node > Impact**.
The **Impact Analysis Report** window opens:

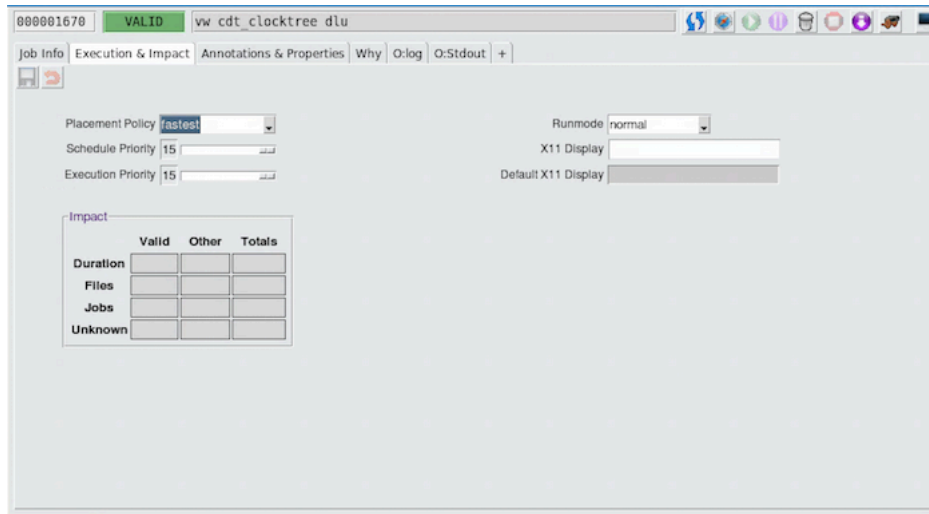


Figure 18:

The impact analysis shows how many files and jobs are affected by a change in the selected file. Because FlowTracer keeps a record of the time it has taken to run each job, FlowTracer can predict the time it will take to execute all jobs dependent on the selected file.

You can get the same information from the command line with the command `vsc`:

```
% vsc aa
VALID NODES Files: 4    Tools:    4    Duration:1s
OTHER NODES Nothing.
-----
TOTAL Files:    4 Tools:    4    Duration:1s
```

Forget Nodes and Sets from the Graph

FlowTracer remembers the jobs you execute provided you enable runtime tracing for those jobs, as you have done in this tutorial by using the FlowTracer wrapper `vw`. It often becomes necessary to tell FlowTracer to forget about parts of a flow.

1. To forget a single node:
 - a) Point at the node.
 - b) Right-click, hold, and select **Forget**.
2. To forget multiple nodes:
 - a) Select the nodes you want to forget by drawing a rubber band around them.
 - b) Point at one of the selected nodes.
 - c) Right-click, hold, and select **Forget** from the pop-up menu.
3. To forget a set (not the elements in the set):
 - a) Select the set in the Set Browser.
 - b) Look at the sets in the "Tmp" folder. From the menu, select **Set Forget Set Only**.



Note: System sets cannot be forgotten. "Predefined" sets can be forgotten but they will not be removed from the set hierarchy. Double-clicking on any of the "Predefined" set names will automatically recreate that set.

- c) You can also choose to forget a set and all the contents (nodes) of that set by using the option **Forget Set & Elements** instead.

Command Line Interface

Everything that you have done with the console, you can do from the command line. We recommend that you keep up the console, for now, so you can monitor the effect of your actions. As you learn the Command Line Interface (CLI) programs, you will be able to use the CLI or the GUI.

In this tutorial we cover the most important commands in FlowTracer. For a complete list of commands, check the Global Commands List.

Check File Status

1. The command `vls` can be used to check the status of files. First try it when all files are VALID.

```
% vls
VALID i aa
VALID u bb
VALID u cc
VALID o dd1
VALID o dd2
```

The first column shows the status of the files in the context of the flow. The second column summarizes the connectivity information for the file: "i" indicates a primary input, "o" indicates a primary output, "u" indicates files that have both inputs and outputs.

2. Now, change aa and check again, this time using the option `-l` to get more information.

```
% touch aa
% vls -l
1 00000582 i VALID aa
3 00000013 u INVALID bb
5 00000016 u INVALID cc
7 00000193 o INVALID dd1
7 00000265 o INVALID dd2
```

With the `-l` option you can see two more columns. The first column shows the level of the node in the graph. Level 1 is at the top. The second column contains the VovId of the files, a unique identifier used in most FlowTracer operations.

3. Use option `-h` or `-help` to get a help message.

Check Job Status

The `vst` command can be used to check on job status.

1. Use the command `vst` to check the status of jobs. The letter "t" stands for "tool invocation" which is a synonym for job.

```
% vst
00000638 INVALID vw cp aa bb
00000689 INVALID vw cp bb cc
00000729 INVALID vw cp cc dd2
00000749 INVALID vw cp cc dd1
```

This command also supports many options, which you can see using the option `-h` or `-help`. Important is the option `-a`, which

2. Add the `-h` or `- help` option to see what other options are supported.
3. Add the `-a` option to show the environment in which the jobs have been executed:

```
% vst -a
vst: rule is `ISJOB==1 CWD==${HOME}/simple_test'
vst: format is `@LEVEL:3@ @ID@ @STATUS:10@ @ENV:8@ @COMMAND@'
2 00000638 INVALID BASE vw cp aa bb
4 00000689 INVALID BASE vw cp bb cc
6 00000729 INVALID BASE vw cp cc dd2
6 00000749 INVALID BASE vw cp cc dd1
```

Rerun from the CLI

The command `vsr` is used to issue rerun requests. The target to update can be a file, a directory, a set, or a list of files directories and sets.

Try the following commands:

```
% touch aa
% vsr bb
-----
| Retrace : tmp:retrace:to ${HOME}/simple_test/bb
| Id : 00000848
| Requested by: john@tahoe:0.0
| Priority : NORMAL
| Mode : SAFE
| Work to do : 1 tools
| Status : Completing in: 1s.
|-----
beatty<-- vw cp aa bb
-----
| Retrace : tmp:retrace:to ${HOME}/simple_test/bb
| Id : 00000848
| Requested by: john@tahoe:0.0
| Priority : NORMAL
| Mode : SAFE
| Work to do : 1 tools
| Status : DONE. Expected duration: 1s Actual: 1s (100%)
|-----
% vsr .
-----
| Retrace : tmp:retrace:dir ${HOME}/simple_test
| Id : 00001219
```

```
| Requested by: john@tahoe:0.0  
| Priority : NORMAL  
| Mode : SAFE  
| Work to do : 3 tools  
| Status : Completing in: 3s.  
-----  
beatty<-- vw cp bb cc  
beatty<-- vw cp cc dd1  
beatty<-- vw cp cc dd2  
-----  
| Retrace : tmp:retrace:dir ${HOME}/simple_test  
| Id : 00001219  
| Requested by: john@tahoe:0.0  
| Priority : NORMAL  
| Mode : SAFE  
| Work to do : 3 tools  
| Status : DONE. Expected duration: 3s Actual: 10s (333%)  
-----  
% vsr -all  
... output omitted ...
```

Detect Conflicts

If you make no mistakes, FlowTracer remains invisible. However, if in your design activity you accidentally violate dependency constraints, FlowTracer will alert you. FlowTracer will warn you if you try to execute a tool with invalid inputs.

1. Execute the following:

```
% touch aa  
% vw cp bb cc
```

In this case, the modification to aa invalidated bb and therefore made the request to run the job of copying bb to cc a wasted step because bb was no longer valid. When you request that job to be done, FlowTracer will prompt you as follows:

```
FlowTracer: ATTENTION! Input conflict detected! FlowTracer: ATTENTION!  
FlowTracer: ATTENTION! Input conflict detected! FlowTracer: ATTENTION!  
----- User Decision Required -----  
INPUT CONFLICT for tool  
vw cp bb cc  
(directory ${HOME}/tutorial)  
The tool needs  
FILE:${HOME}/tutorial/bb  
which is currently INVALID  
1 -- CONTINUE  
2 -- STOP ASKING  
3 -- (*) ABORT  
Please choose (1--3) >>>
```

Here you have the chance to abort from an operation that has to be redone anyway later, or continue as you would have without FlowTracer. At least you are aware that the computation is likely to be incorrect.

2. If you try to redefine the source of a file, FlowTracer will ask you if you really want to change how the file is generated. Try the following

```
% vw cp aa bb  
% vw cp bb cc  
% vw cp aa cc
```

Example of an output conflict :

```
FlowTracer: ATTENTION! Output conflict detected! FlowTracer: ATTENTION!  
FlowTracer: ATTENTION! Output conflict detected! FlowTracer: ATTENTION!  
----- User Decision Required -----  
OUTPUT CONFLICT caused by data item  
FILE:${HOME}/tutorial/cc  
Command lines are different.  
Common part is 6 characters long.  
'bb cc' != 'aa cc'.  
^      ^  
1 -- CONTINUE  
2 -- (*) ABORT  
3 -- MORE INFO  
Please choose (1--3) >>>
```

You can see that cc is already dependent on bb.

3. Answer 2 (Abort).

FlowTracer will also prevent you from creating cyclic dependencies.

```
% vw cp aa bb  
% vw cp bb cc  
% vw cp cc aa # whoops, a cycle (aa->bb->cc->aa)
```

Example of an cycle conflict :

```
FlowTracer: ATTENTION! Cycle detected! FlowTracer: ATTENTION!  
FlowTracer: ATTENTION! Cycle detected! FlowTracer: ATTENTION!  
vw Mar 30 12:36:03 Failed FlowTracer call libconnect.cc,146  
  
vw ERROR Mar 30 12:36:03 Cycle conflict for ${HOME}/tutorial/aa  
vw Mar 30 12:36:03 This tool invocation is now forgotten  
vw Mar 30 12:36:03 Serious dependency violation (status -3)
```

Repeat the Tutorial without the GUI

Rerunning in the current directory is initiated with the command `vsr`. Right now, the dependency graph should be up to date with all nodes being valid (you can check this status using `vls`). So running `vsr` will do nothing.

1. Run everything with the `vsr` command:

```
% vsr  
-----  
| Retrace : Retrace Directory ...  
-----  
sparc<-- vw cp aa bb  
hppa <-- vw cp bb cc  
sparc<-- vw tar -cf archive.tar aa bb cc  
-----  
| Status : DONE. Expected duration: 2s Actual: 3s (150%)  
-----
```

If FlowTracer has been configured to use multiple machines, you may see commands being executed on other hosts. FlowTracer uses a technique called resource mapping to generate a list of candidate machines, then selects the machine which can execute the command the fastest.

2. Run the command `vls` to confirm that the system has been updated.

```
% vls  
VALID i aa  
VALID o archive.tar  
VALID u bb  
VALID u cc
```

Flow Description Language

In the earlier section, you built a flow by interactively calling the wrapper program for each job. That was a useful exercise to understand a simple way to register a job with FlowTracer, how to establish runtime tracing for the job, and how to monitor a flow using the GUI console. However, that is not how FlowTracer would be used for production.

The FlowTracer product does not expect a product flow to be registered by interactively registering each job one at a time from the command line. Instead, the normal use is:

- Develop a description of the jobs to register, using the Flow Description Language (FDL)
- Register the jobs and instantiate the flow using the `vovbuild` command
- Request a `"run"` to have FlowTracer schedule and deploy jobs as necessary in parallel

In this tutorial, you will develop a few simple flows by writing a job description file and registering the jobs with `vovbuild`.

Tasks in This Tutorial

Remove Older Sets

Before starting, you should establish your current working directory to be the same one you used earlier "simple_test" in your home, and you should have the GUI Console running so you can watch the effect of building the flow, as you did in the earlier tutorial.

1. Change to your current working directory and start up `vovconsole` in the background.

```
% cd
% cd simple_test
% vovconsole &
```

2. From the Set Browser, click on **System** and then double click on **Nodes**.
3. Double click on a set name to display the contents of the set in the Set Viewer panel on the right.
4. Single click on a set name to highlight nodes in the Set Viewer panel if they are members of the clicked set. For example, if a set named **TOP:partition1:subset1** is clicked while showing **TOP:partition1** in the Set Viewer panel, then the contents of subset1 will be highlighted in the Set Viewer. You should see the current state of the flow from when you stopped the earlier tutorial. The model of the flow is held in the server, not the console. The console shows what the server is managing.
5. The view of the graph can be toggled to show or not show files. We want to have the files show. If the files are not shown, turn on display of file nodes by right clicking in the background of the Set Viewer panel to open a context menu. Click **Show/Hide** to open a submenu where you can toggle on the **Show Files** option.

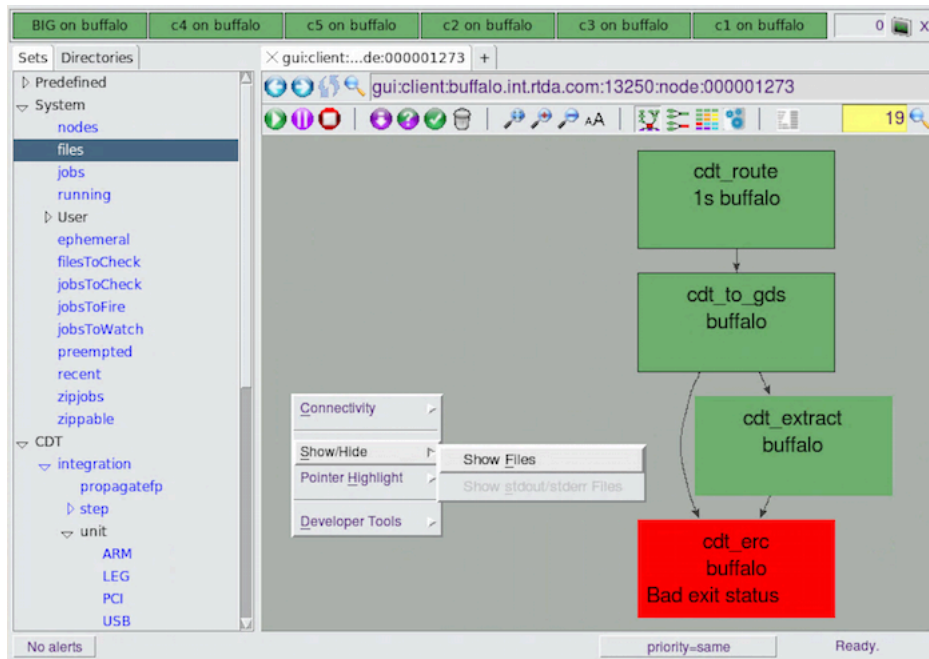


Figure 19: Context menu seen when right-clicking in open area of canvas in Set Viewer

For the next exercise, you will not be using the flow from the previous steps. You can tell FlowTracer sever to drop that flow from its memory. This will remove it from the server, and the console display will change to show that the flow was dropped.

You tell the server to forget by telling it to forget nodes. You can tell it to forget one node or groups of nodes using the `vovforget` command. The easy way to refer to a group of nodes is by referencing them by set name.

6. Tell FlowTracer to forget the nodes you see when displaying the **System** > **nodes** set by telling it to forget the nodes in that set. Enter this, typing it in at the command line:

```
% vovforget -elements System:nodes
message: Forgotten 10 nodes
```

The Set Viewer display updates to show an empty canvas since there are no longer any nodes to display, as they have all been forgotten.

7. Remove the files from the directory `simple_test`. It will contain the file `aa` which is the primary input of the emulation, and the files `bb`, `cc`, `dd1`, and `dd2` which are the derived files from the emulation. Enter the following:

```
% rm aa bb cc dd1 dd2
```

You are now ready to continue with this exercise to register a new set of jobs that define a flow by way of editing a Flow Description file and using it to build the flow.

The Flow.tcl File

The default name for a job description file is `Flow.tcl`. The first job description file you will write will define the same jobs that you registered interactively in the previous tutorial.

You should have a file called `Flow.tcl` in the `simple_test` directory. It will hold a Flow Description for the flow having four jobs that was built interactively before. Recall that this tutorial is using the `cp` command to emulate more complicated programs that process input to generate output.

Edit the file as follows:

```
% cat Flow.tcl
J vw cp bb cc
J vw cp cc dd1
J vw cp cc dd2
J vw cp aa bb
```

The sequence of the commands is deliberately in the "wrong" order from how you might enter them if you planned to run the programs yourself. You do not need to enter them in their dependency order.

The token 'J' in this file is the name of a Tcl procedure, one of those that comprise the Flow Description Language. J means to register a job into the flow. The job to register is the one whose command has been passed as an argument, (the command is what follows on the line). The command is what was typed in interactively in the earlier tutorial. The command calls the wrapper program "vw" to establish that runtime tracing will be used. It passes the wrapper the shell command that runs the job.

In this case we want our flow to contain 4 jobs. The jobs are in an arbitrary order, since FlowTracer has the ability to determine or discover the correct order.

The flow description can be this simple because it need not be concerned with issues like environment setup, job scheduling, job control, capturing of stdout and stderr, license checking, error checking, detection of parallelism, since all these services are automatically provided by FlowTracer. This means that a flow description file is typically several times smaller than an equivalent Makefile or shell script.

Build the Flow

Now that you have a job description, you need to build the flow with the `vovbuild` program.

The `vovbuild` program processes a flow description file and registers the described programs into the flow. It defaults to using the file `Flow.tcl` as the flow description file.

```
% vovbuild
.... # 4 dots, one per job
```

Building the flow is different from running it. The jobs in the flow may take seconds or days to execute, but building the flow is normally a rather quick step. Building the flow is building the dependency graph, not running the programs registered into the graph.

After the `vovbuild` is done, you must double click the **System:nodes** set in the Set Browser to get the console to refresh the Set Viewer with the current graph.

The graph you get will have a similar look to this. Minor differences in the size and position of the nodes are to be expected.

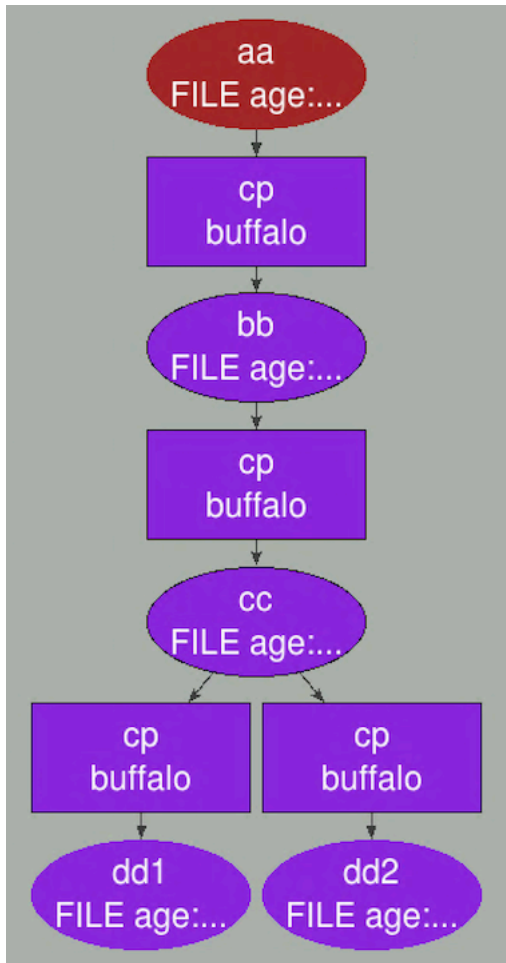


Figure 20:

Notice that the dependency ordering of the jobs in the graph has been created properly, even though the jobs in the FDL file were entered without an order.

Notice that the top node is brown, to indicate that file aa is missing. The flow building discovered that file aa is not present and marks the file to have the MISSING status. The other nodes are purple to indicate that they are INVALID.

Run the Flow Interactively

1. You can run the flow by pressing **Run** in the action menu bar in the top of the Set Viewer panel. Nothing will happen since the very first job in the dependency graph is unable to run since its input aa is missing.
2. Create a file aa in the simple_test directory.

```
% touch aa
```

The graph will change to show that file aa was just created. If this is not seen, make sure that the Set Viewer is actively displaying the **System:nodes** set by double clicking on the **System:nodes** choice in the Set Browser.

The node for file aa is green. The rest of the nodes are purple to indicate that all those dependent elements are INVALID. Another way to see the INVALID set of nodes is to say they are ready to run.

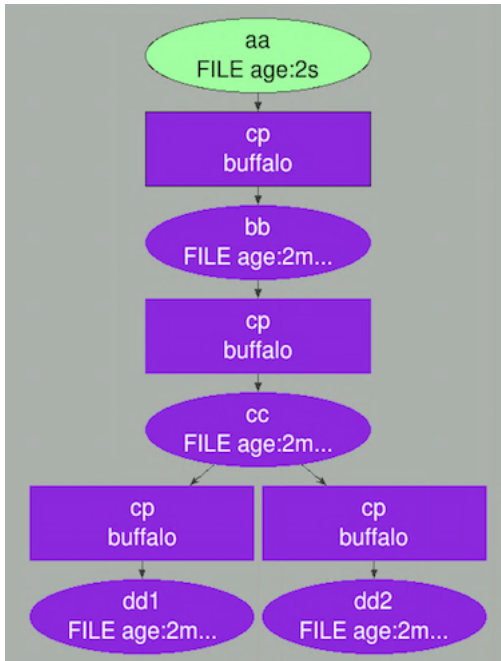


Figure 21:

3. Click **Run** to have FlowTracer process the graph.

The display will change quickly to reflect the various state changes of the nodes representing the jobs and the files.

You will see nodes change to light blue to indicate they are SCHEDULED, or yellow to indicate they are RUNNING, and finally, all of the nodes will turn bright green to show a successful run of the dependency graph. All the dependent jobs ran successfully and all dependent files were made successfully. This is just an emulation using the `cp` command but if the defined jobs had used production class programs, the result would be the same.

Running a job interactively is fun to do but it is not how FlowTracer would be used in production. This section was to show you how the flow description language file is created and how the flow is registered with FlowTracer by using `vovbuild`.

Run a Flow from the Command Line

In production mode, you will use a command line request to run the flow. It's as easy as clicking the **Run** button.

1. Now that there is a `Flow.tcl` file in hand that can be given to `vovbuild` to create a flow as needed, remove the current flow and register it again. Remove the dependent files too, leaving the primary file `aa`.

```
% vovforget -elements System:nodes
message: Forgotten 10 nodes
% rm bb cc dd1 dd2
% vovbuild
....
```

The GUI Console will have the familiar look to it as seen earlier. Note that there may be slight variation in the placement of the nodes in the display, which is expected.

2. Enlarge the display and select the menu **View > Fit** or use the keyboard shortcut **f** to cause the placement of nodes to fit better within the Set Viewer.
3. Resize the display smaller, which will keep the same layout but reduce the size of the nodes.
4. Use the command line to request that FlowTracer run the jobs, taking into account the dependencies of the graph, just as it does when the run request is done by clicking the Run button. The program `vsr` is the command that requests FlowTracer to run the flow.

```
% vsr
....
localhost <-- vw cp aa bb
localhost <-- vw cp bb cc
localhost <-- vw cp cc dd
localhost <-- vw cp cc ee
```

The resulting graph in the GUI Console will show various changes in node colors as the dependent programs are run in the right order, and dependent files are created. In the end, the graph will show all green nodes, indicating a successful run.

Batch Process to Define and Run a Flow

In production mode, you will have scripts that create flows and run them. The console GUI will be one interface you will use to monitor the flows that get run this way. The other interface is through the browser.

In these tutorials, you are using the simple command `cp` to emulate more complicated programs, which might take much longer to run than a file copy does. This tutorial does not show you long running programs, as you will see in production. This means that during this exercise you do not get to view long lasting states of an intermediate job. The intermediate jobs run too quickly to notice their state changes. For longer lasting flows, the GUI and browser interface provide useful ways to watch the details of what is happening.

1. The control over what is supposed to happen is in scripts that define the flows and get them running. To demonstrate this, reset FlowTracer as you did earlier, to get rid of the current flow. It is not needed any more. Remove the dependent files and the primary input file too.

```
% vovforget -elements System:nodes
message: Forgotten 10 nodes
% rm aa bb cc dd1 dd2
```

The console should reflect all this and show an empty Set View panel.

2. Edit the file `dowork.sh` in the `simple_test` directory to look like this:

```
% cat dowork.sh

rm -f aa bb cc dd1 dd2
vovforget -elements System:nodes
sleep 5
vovbuild -f Flow.tcl
sleep 5
vsr
sleep 10
```

```
touch aa  
sleep 10  
vsr
```

This shell script is going to run the commands you have been entering interactively, with sleep commands in the sequence to emulate the time between entering the commands. This is intended to give you a chance to notice the state change in the console's Set Viewer when the script is run.

3. Run the shell script while watching the GUI console.

```
% sh dowork.sh
```

The script starts with removing the primary and dependent files so it can be run repeatedly.

4. Run the shell script again, and then again, to get familiar with having flows defined starting from an empty canvas, seeing the flow dependency graph before any jobs of the flow are run, seeing the jobs scheduled to run but not succeed, and then seeing a successful run of all the jobs.

The flow is defined by FDL language statements in a flow description file. The flow is created and run by commands in a batch shell script. You monitor the progress of the entire system by looking at the GUI console.

Create a Complex Flow

Everything done so far could have been done just as easily using *make* or a C-shell script. In this step you will build a flow which neither *make* nor a shell script could handle efficiently. This is a flow that spans multiple directories.

This example project will dynamically create a set of subdirectories. For each subdirectory, it will run four jobs within that context that process the same input file *aa* that comes from the top level main directory. The jobs are the ones we have been using to emulate useful work.

This flow definition implements a project that has variant ways of processing, starting from a given input file, with each variant task branch running in a different area, and each one enabled to be run in parallel or in any order, independent of work done in another subdirectory. For this tutorial flow, all the variant task branches have the same set of four jobs which are emulated by the same simple *cp* commands. In a real project, each variant task sequence could involve different programs.

```
% cat Flow2.tcl  
  
for {set i 1} { $i < 20 } { incr i } {  
  indir -create subdir$i {  
    J vw cp ../aa bb  
    J vw cp bb cc  
    J vw cp cc dd1  
    J vw cp cc dd2  
  }  
}
```

The *for* construct is standard Tcl, while the procedure *indir* is a FlowTracer extension. In this case you want to create the subdirectories *subdirN*, so you will use the option *-create* of *indir*.

1. Start the sequence with removal of files that might exist if the steps are done again.

```
% vovforget -elements System:nodes  
% rm -rf aa subdir*  
% vovbuild -f Flow2.tcl  
.....
```



```
.....  
.....  
.....  
% touch aa  
% vsr -all
```

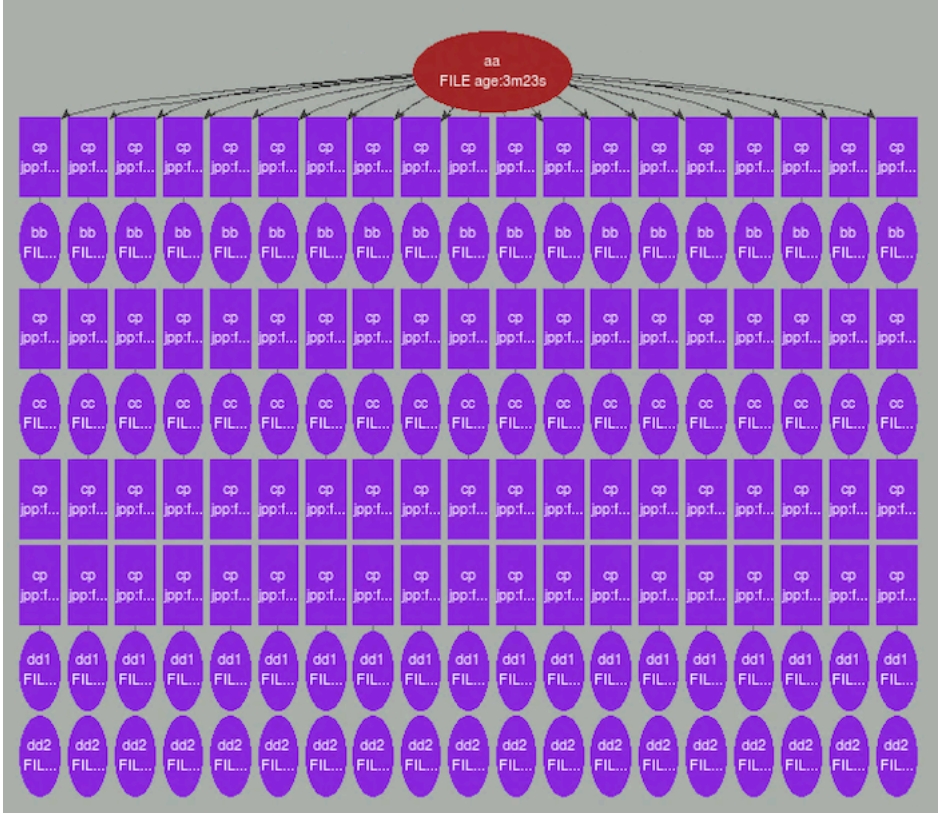


Figure 22:

You have to use option -all of vsr because this flow spans multiple directories and the default target of vsr is just the current working directory.

- 2. As was done earlier, create a batch shell script to build the flow and run it.

```
% cat dowork2.sh  
  
vovforget -elements System:nodes  
rm -rf aa subdirs*  
sleep 5  
vovbuild -f Flow2.tcl  
sleep 5  
touch aa  
sleep 5  
vsr -all  
  
% sh dowork2.sh
```

- 3. Rather than use a shell script to redo the commands to build and run the flow, you can work with the flow as it is defined in FlowTracer, and try touching aa to emulate a change in the primary input file. This models an event that precedes running all the dependent jobs. Run vsr -all to have FlowTracer schedule and dispatch all the dependent variant processes in the task paths. Watch the state of the nodes as the activity progresses.

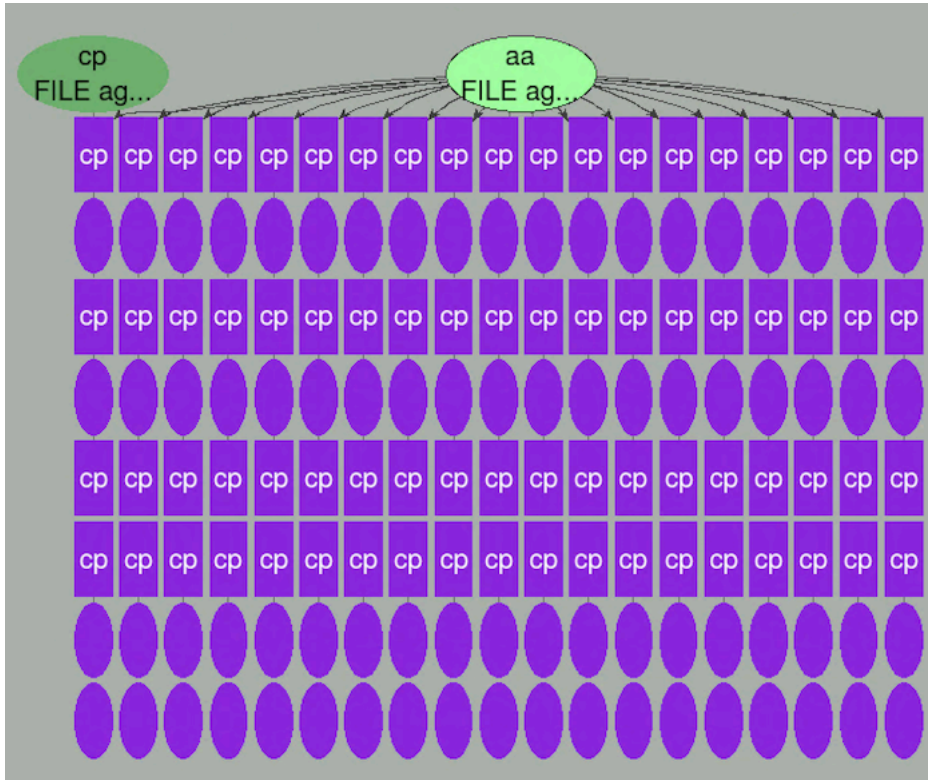


Figure 23:

EDA Flows

In this section, some typical EDA flows that are more dynamic than the ones we have used so far are shown.

You will consider a project that implements a simulation based on calling a tool named `simulate`. The model is that a directory will contain a group of stimulus files. The intent is to run one job for each stimulus file in the directory. The job will run the `simulate` program within the context of a unique subdirectory for each stimulus file.

The Tcl file to define the jobs in this project will use the Tcl `glob` notation to discover all the stimulus files in the directory based on their having a name with the suffix `.stim`. Then it will create a subdirectory using the base name of the stimulus file, and run a job in that subdirectory.

In this example, two FDL procedures are introduced: `E` and `R`. The procedure `E` defines the environment in which the simulation jobs must be executed. In this case, the environment is the combination of the `BASE` environment, which is part of any normal FlowTracer installation, and the `SPICE` environment, which is presumably an environment that has to be setup for each site to support the running of the `SPICE` tool, since the location of the simulation software varies from site to site.

The procedure `R` defines the resources required by the subsequent jobs in the flow. In this case, we declare that each job requires one license of the tool 'spice' (represented by the resource 'License:spice') and at least 250MB of RAM.

This is an example of how to define a group of simulation jobs for a project. These lines would be placed into a tcl file so that the tcl file could be registered into FlowTracer by running *vovbuild* against it.

```
E "BASE+SPICE"  
R "License:spice RAM/250"  
foreach stimulusFile [glob *.stim] {  
    set root [file root $stimulusFile]  
    indir -create $root {  
        J vw simulate ../$stimulusFile -o $root.log  
    }  
}
```

This shows how jobs would be defined in a production environment so that a project's flow gets defined by way of running *vovbuild* against a script holding the job definitions using the FDL language and tcl.

The back-end flows for placement and routing of blocks tend to require many sequential steps, each one requiring different resources, such as licenses and RAM. While many organization use the same tool suites, such as Cadence's Silicon Ensemble, it is rare to see the core tools such as *qp* and *wroute* called directly. Instead, each organization has its own wrapper script to define how those tools are to be invoked. In our example, the wrapper script is called *pnr* and is presumably accessible from the environment called EDA.

Example of defining a group of jobs to do a Place & Route operation:

```
set block [shift]  
E "BASE+EDA+CADENCE"  
  
R "License:qp RAM/250"  
J vw pnr place $block  
J vw pnr scanins $block  
  
R "License:wroute RAM/2000"  
J vw pnr route $block  
J vw pnr clocktree $block  
  
R ""  
J vw pnr to_gds $block
```

Stop the Project

Server Management: Starting and Stopping

The server normally runs for the lifetime of the project. If it becomes necessary to shut down the server, use the stop option of the `vovproject` command.

```
% vovproject stop
vovproject mm/dd/2015 hh:mm:ss: message: Checking privilege to stop project 'test'
Shut down test (yes/no)? yes
...
```

You can later restart the server with

```
% vovproject start project
```

Server Management: Destroying

After stopping a project, you can completely remove the project using the destroy option of the `vovproject` command.

```
% vovproject destroy project
```

This command removes all project files from the file system so that the project does not exist anymore and can not be started.

This chapter covers the following:

- [Create Efficient VOV Scripts](#) (p. 46)
- [Write Flows](#) (p. 48)

Create Efficient VOV Scripts

If your flows are small, such as a few thousands jobs, you probably do not need to worry much about efficiency of your scripts. If you expect to operate on flows with hundreds of thousands of jobs, then this section can be useful.

While developing a VOV script, it is important to make sure that they do not needlessly make expensive calls that take a lot of vovserver time.

One useful method is to ask the system to show the service time for all expensive calls, which is activated by setting the environment variable `VOV_SHOW_SERVICE_TIME` to a positive integer that represents a time in milliseconds.

 **Note:** The integer value is a threshold below which the times are not shown.

Here is an example with a call (i.e. "sanity") that tends to be expensive:

```
% setenv VOV_SHOW_SERVICE_TIME 1
% vovproject sanity
vovsh(19194) Nov 12 12:36:35 SERVICE_TIME: Service took 2027ms for 137=SanityCheck
vovsh(19194) Nov 12 12:36:35 SERVICE_TIME: Total service time for this client:
2.027s
```

In this example, vovserver took a bit more than 2 seconds to complete the reply to the request "SanityCheck" (internal code 137). This is normal for SanityCheck, and it is a reason why you do not want to run SanityCheck unless really necessary. Most VOV services you really need should be in the low millisecond range.

This method only shows the "slow" services. To see all services requested by a script, use the variable `VOV_DEBUG_FLAGS` as in this example:

```
% setenv VOV_DEBUG_FLAGS 16 ; ### This has to be 16 to show the RPC codes.
```

Experiments

```
#!/bin/csh -f
# Try this script and compare the load on the server
# Assume it is called "my_test_script"

set id = `vovsh -x 'FDL_INIT; VovUtils:init; set vovutils(feedback) quiet; puts [J vw
hostname]`
vovselect status from jobs where id==$id ; ##### A common mistake
vovselect status from $id

# NC variants
nc info $id | grep Status | awk '{print $2}'
nc list | grep $id | awk '{print $2}' ;;; ## Another horrible yet common mistake
nc getfield $id status ;;; ## BEST way!

##### NOTE: This experiment run with 500,000 jobs in the flow.
% setenv VOV_SHOW_SERVICE_TIME 1
% unsetenv VOV_DEBUG_FLAGS
% ./my_test_script
vovsh(9612) Nov 12 15:19:48 SERVICE_TIME: Total service time for this client:
0.000s
vovsh(9617) Nov 12 15:19:48 SERVICE_TIME: Service took 8ms for 307=CreateQuery
select:fieldname from:jobs
```

```
vovsh(9617) Nov 12 15:19:48 SERVICE_TIME: Service took 707ms for 307=CreateQuery
select:status from:jobs where:id==002233767
INVALID
vovsh(9617) Nov 12 15:19:48 SERVICE_TIME: Total service time for this client:
0.715s
INVALID
vovsh(9671) Nov 12 15:19:49 SERVICE_TIME: Total service time for this client:
0.000s
vovsh(9698) Nov 12 15:19:49 SERVICE_TIME: Service took 1ms for 208=GetInfoMap
project
vovsh(9698) Nov 12 15:19:49 SERVICE_TIME: Total service time for this client:
0.001s
Idle
vovsh(9724) Nov 12 15:19:52 SERVICE_TIME: Service took 2607ms for
296=ListElementsEnh id:000001041 format:@ID@ @STATUSNC:9@ @PRIORITYPP:6@ @HOST:14@
@COMMAND:40@ range:0--1
vovsh(9724) Nov 12 15:19:54 SERVICE_TIME: Total service time for this client:
2.607s
Idle
INVALID
vovsh(9928) Nov 12 15:19:55 SERVICE_TIME: Total service time for this client:
0.000s
```

Write Flows

In [Create a FlowTracer Project](#), you built a flow by executing one tool at a time. That was a useful exercise to understand the fundamentals of runtime tracing. However, that is not the usage model for FlowTracer.

The flow developer rarely has to enter any shell command. In the normal usage of FlowTracer, developers use the FDL and tool integration to build flows

In this tutorial, you will write a few simple flows.

The Flow.tcl file

The normal name for a flow description is `Flow.tcl`. The first flow you will write will reproduce the flow created in the user tutorial:

```
# This is the first Flow.tcl
J vw cp bb cc
J vw cp cc dd
J vw cp cc ee
J vw cp aa bb ;# Deliberately out of order.
```

The token `J` in this file is the name of a Tcl procedure, one of those that comprise the Flow Description Language. `J` means that we want our flow to include the job whose command line has been passed as argument.

In this case we want our flow to contain 4 jobs. We list the jobs in arbitrary order, since FlowTracer has the ability to determine or discover the correct order anyway.

The flow description can be this simple because it need not be concerned with issues like environment setup, job scheduling, job control, capturing of stdout and stderr, license checking, error checking, detection of parallelism, since all these services are automatically provided by FlowTracer. This means that a flow description file is typically several times smaller than an equivalent Makefile or shell script.

Build the Flow

Now that we have a flow description, we need to build the flow with `vovbuild`. Before we do that, however, we recommend that you use the GUI to monitor what is happening, as you have learned in the user tutorial. Also, in case you still have the flow generated in the user tutorial, you should tell FlowTracer to forget it.

```
% vovconsole &
% vovforget -elements System:nodes
% vovbuild
.... # 4 dots, one per job.
```

Building the flow is different from running it. The jobs in the flow may take hours or days to execute, but building the flow is normally a rather quick step.

Execute the Flow

Now you can ask FlowTracer to run the jobs for you, taking into account dependencies and parallelism.

```
% vsr
....
localhost <-- vw cp aa bb
localhost <-- vw cp bb cc
localhost <-- vw cp cc dd
localhost <-- vw cp cc ee
```

Build a More Complex Flow

Everything done so far could have been done just as easily using make or a C-shell script. In this step we build a flow which neither make nor a shell script could handle efficiently. This is a flow that spans multiple directories.

```
# This is Flow2.tcl
for {set i 1} { $i < 20 } { incr i } {
  indir -create subdir$i {
    J vw cp ../aa bb
    J vw cp bb cc
    J vw cp cc dd
    J vw cp cc ee
  }
}
```

The for construct is standard Tcl, while the procedure indir is a FlowTracer extension. In this case we want to create the subdirectories `subdirN`, so we use the option `-create` of `indir`.

```
% vovbuild -f Flow2.tcl
.....
.....
.....
.....
.....
% vsr -all
```

You have to use option `-all` of `vsr` because this flow spans multiple directories and the default target of `vsr` is just the current working directory.

EDA Flows

In this section we show some typical EDA flows. We start with a simulation flow, where we want to execute one job for each stimulus file in a directory. We use `glob` to find all stimuli, that is, the files with suffix `".stim"`, then we create a subdirectory for each file and we define a job to run in such directory.

In this example we introduce two FDL procedures: `E` and `R`. The procedure `E` defines the environment in which the simulation jobs must be executed. In this case, the environment is the combination of the `BASE` environment, which is part of any normal FlowTracer installation, and the `SPICE` environment, which is presumably an environment that has to be setup for each site, since the location of the simulation software varies from site to site.

The procedure R defines the resources required by the subsequent jobs in the flow. In this case, we declare that each job requires one license of the tool 'spice' (represented by the resource 'License:spice') and at least 250MB of RAM.

A simulation flow:

```
E "BASE+SPICE"  
R "License:spice RAM/250"  
foreach stimulusFile [glob *.stim] {  
  set root [file root $stimulusFile]  
  indir -create $root {  
    J vw simulate ../$stimulusFile -o $root.log  
  }  
}
```

The back-end flows for placement and routing of blocks tend to require many sequential steps, each one requiring different resources, such as licenses and RAM. While many organizations use the same tool suites, such as Cadence's Silicon Ensemble, it is rare to see the core tools such as qp and wroute called directly. Instead, each organization has its own wrapper script to define how those tools are to be invoked. In our example, the wrapper script is called pnr and is presumably accessible from the environment called EDA.

A Place & Route flow

```
set block [shift]  
E "BASE+EDA+CADENCE"  
  
R "License:qp RAM/250"  
J vw pnr place $block  
J vw pnr scanins $block  
  
R "License:wroute RAM/2000"  
J vw pnr route $block  
J vw pnr clocktree $block  
  
R ""  
J vw pnr to_gds $block
```

Legal Notices

Intellectual Property Rights Notice

Copyrights, trademarks, trade secrets, patents and third party software licenses.

Copyright © 1986-2023 Altair Engineering Inc. All Rights Reserved.

This Intellectual Property Rights Notice is exemplary, and therefore not exhaustive, of intellectual property rights held by Altair Engineering Inc. or its affiliates. Software, other products, and materials of Altair Engineering Inc. or its affiliates are protected under laws of the United States and laws of other jurisdictions. In addition to intellectual property rights indicated herein, such software, other products, and materials of Altair Engineering Inc. or its affiliates may be further protected by patents, additional copyrights, additional trademarks, trade secrets, and additional other intellectual property rights. For avoidance of doubt, copyright notice does not imply publication. Copyrights in the below are held by Altair Engineering Inc. or its affiliates. Additionally, all non-Altair marks are the property of their respective owners.

This Intellectual Property Rights Notice does not give you any right to any product, such as software, or underlying intellectual property rights of Altair Engineering Inc. or its affiliates. Usage, for example, of software of Altair Engineering Inc. or its affiliates is governed by and dependent on a valid license agreement.

Altair Simulation Products

Altair® AcuSolve® ©1997-2023

Altair Activate® ©1989-2023

Altair® Battery Designer™ ©2019-2023

Altair Compose® ©2007-2023

Altair® ConnectMe™ ©2014-2023

Altair® EDEM™ © 2005-2023

Altair® ElectroFlo™ ©1992-2023

Altair Embed® ©1989-2023

Altair Embed® SE ©1989-2023

Altair Embed®/Digital Power Designer ©2012-2023

Altair Embed® Viewer ©1996-2023

Altair® ESAComp® ©1992-2023

Altair® Feko® ©1999-2023

Altair® Flow Simulator™ ©2016-2023

Altair® Flux® ©1983-2023

Altair® FluxMotor® ©2017-2023

Altair® HyperCrash® ©2001-2023

Altair® HyperGraph® ©1995-2023

Altair® HyperLife® ©1990-2023

Altair® HyperMesh® ©1990-2023
Altair® HyperSpice™ ©2017-2023
Altair® HyperStudy® ©1999-2023
Altair® HyperView® ©1999-2023
Altair® HyperViewPlayer® © 2022-2023
Altair® HyperWorks® ©1990-2023
Altair® HyperXtrude® ©1999-2023
Altair® Inspire™ ©2009-2023
Altair® Inspire™ Cast ©2011-2023
Altair® Inspire™ Extrude Metal ©1996-2023
Altair® Inspire™ Extrude Polymer ©1996-2023
Altair® Inspire™ Form ©1998-2023
Altair® Inspire™ Mold ©2009-2023
Altair® Inspire™ PolyFoam ©2009-2023
Altair® Inspire™ Print3D ©2021-2023
Altair® Inspire™ Render©1993-2023
Altair® Inspire™ Studio ©1993-2023
Altair® Material Data Center™ ©2019-2023
Altair® MotionSolve® ©2002-2023
Altair® MotionView® ©1993-2023
Altair® Multiscale Designer® ©2011-2023
Altair® nanoFluidX® ©2013-2023
Altair® OptiStruct® ©1996-2023
Altair® PolEx™ ©2003-2023
Altair® PSIM™ © 2022-2023
Altair® Pulse™ ©2020-2023
Altair® Radioss® ©1986-2023
Altair® romAI™ © 2022-2023
Altair® S-FRAME® © 1995-2023
Altair® S-STEEL™ © 1995-2023
Altair® S-PAD™ © 1995-2023
Altair® S-CONCRETE™ © 1995-2023
Altair® S-LINE™ © 1995-2023

Altair® S-TIMBER™ © 1995-2023

Altair® S-FOUNDATION™ © 1995-2023

Altair® S-CALC™ © 1995-2023

Altair® S-VIEW™ © 1995-2023

Altair® Structural Office™ © 2022-2023

Altair® SEAM® © 1985-2023

Altair® SimLab® © 2004-2023

Altair® SimLab® ST © 2019-2023

Altair SimSolid® © 2015-2023

Altair® ultraFluidX® © 2010-2023

Altair® Virtual Wind Tunnel™ © 2012-2023

Altair® WinProp™ © 2000-2023

Altair® WRAP™ © 1998-2023

Altair® GateVision PRO™ © 2002-2023

Altair® RTLvision PRO™ © 2002-2023

Altair® SpiceVision PRO™ © 2002-2023

Altair® StarVision PRO™ © 2002-2023

Altair® EEvision™ © 2018-2023

Altair Packaged Solution Offerings (PSOs)

Altair® Automated Reporting Director™ © 2008-2022

Altair® e-Motor Director™ © 2019-2023

Altair® Geomechanics Director™ © 2011-2022

Altair® Impact Simulation Director™ © 2010-2022

Altair® Model Mesher Director™ © 2010-2023

Altair® NVH Director™ © 2010-2023

Altair® NVH Full Vehicle™ © 2022-2023

Altair® NVH Standard™ © 2022-2023

Altair® Squeak and Rattle Director™ © 2012-2023

Altair® Virtual Gauge Director™ © 2012-2023

Altair® Weld Certification Director™ © 2014-2023

Altair® Multi-Disciplinary Optimization Director™ © 2012-2023

Altair HPC & Cloud Products

Altair® PBS Professional® © 1994-2023

Altair® PBS Works™ © 2022-2023

Altair® Control™ ©2008-2023

Altair® Access™ ©2008-2023

Altair® Accelerator™ ©1995-2023

Altair® Accelerator™ Plus ©1995-2023

Altair® FlowTracer™ ©1995-2023

Altair® Allocator™ ©1995-2023

Altair® Monitor™ ©1995-2023

Altair® Hero™ ©1995-2023

Altair® Software Asset Optimization (SAO) ©2007-2023

Altair Mistral™ ©2022-2023

Altair® Grid Engine® ©2001, 2011-2023

Altair® DesignAI™ ©2022-2023

Altair Breeze™ ©2022-2023

Altair® NavOps® © 2022-2023

Altair® Unlimited™ © 2022-2023

Altair Data Analytics Products

Altair Analytics Workbench™ © 2002-2023

Altair® Knowledge Studio® © 1994-2023

Altair® Knowledge Studio® for Apache Spark © 1994-2023

Altair® Knowledge Seeker™ © 1994-2023

Altair® Knowledge Hub™ © 2017-2023

Altair® Monarch® © 1996-2023

Altair® Panopticon™ © 2004-2023

Altair® SmartWorks™ © 2021-2023

Altair SLC™ ©2002-2023

Altair SmartWorks Hub™ ©2002-2023

Altair® RapidMiner® © 2001-2023

Altair One™ ©1994-2023

Third Party Software Licenses

AcuConsole contains material licensed from Intelligent Light (www.ilight.com) and used by permission.

For a complete list of Altair Accelerator Third Party Software Licenses, please click [here](#).

Technical Support

Altair provides comprehensive software support via web FAQs, tutorials, training classes, telephone and e-mail.

Altair One Customer Portal

Altair One (<https://altairone.com/>) is Altair's customer portal giving you access to product downloads, Knowledge Base and customer support. We strongly recommend that all users create an Altair One account and use it as their primary means of requesting technical support.

Once your customer portal account is set up, you can directly get to your support page via this link: www.altair.com/customer-support/.

Altair Training Classes

Altair training courses provide a hands-on introduction to our products, focusing on overall functionality. Courses are conducted at our main and regional offices or at your facility. If you are interested in training at your facility, please contact your account manager for more details. If you do not know who your account manager is, e-mail your local support office and your account manager will contact you

Telephone and E-mail

If you are unable to contact Altair support via the customer portal, you may reach out to the technical support desk via phone or e-mail. You can use the following table as a reference to locate the support office for your region.

When contacting Altair support, please specify the product and version number you are using along with a detailed description of the problem. It is beneficial for the support engineer to know what type of workstation, operating system, RAM, and graphics board you have, so please include that in your communication.

Location	Telephone	E-mail
Australia	+61 3 9866 5557 +61 4 1486 0829	anz-pbssupport@altair.com
China	+86 21 6117 1666	pbs@altair.com.cn
France	+33 (0)1 4133 0992	pbssupport@europe.altair.com
Germany	+49 (0)7031 6208 22	pbssupport@europe.altair.com
India	+91 80 66 29 4500 +1 800 208 9234 (Toll Free)	pbs-support@india.altair.com
Italy	+39 800 905595	pbssupport@europe.altair.com
Japan	+81 3 6225 5821	pbs@altairjp.co.jp
Korea	+82 70 4050 9200	support@altair.co.kr

Location	Telephone	E-mail
Malaysia	+91 80 66 29 4500 +1 800 208 9234 (Toll Free)	pbs-support@india.altair.com
North America	+1 248 614 2425	pbssupport@altair.com
Russia	+49 7031 6208 22	pbssupport@europe.altair.com
Scandinavia	+46 (0) 46 460 2828	pbssupport@europe.altair.com
Singapore	+91 80 66 29 4500 +1 800 208 9234 (Toll Free)	pbs-support@india.altair.com
South Africa	+27 21 831 1500	pbssupport@europe.altair.com
South America	+55 11 3884 0414	br_support@altair.com
United Kingdom	+44 (0)1926 468 600	pbssupport@europe.altair.com

See www.altair.com for complete information on Altair, our team and our products.

Index

A

add a job interactively [8](#)
add more jobs to the flow [13](#)
analyze impact [27](#)

B

batch process to define and run a flow [39](#)
build a more complex flow [49](#)
build the flow [36, 48](#)

C

change dependent input file [15](#)
check file status [29](#)
check job status [30](#)
check project information [6](#)
command line interface [29](#)
create a complex flow [40](#)
create a FlowTracer project [4](#)
create a project [5](#)
create a project directory [9](#)
create efficient VOV scripts [46](#)

D

detect conflicts [31](#)
determine reason for invalid node status [27](#)

E

EDA flows [42, 49](#)
enable a shell [6](#)
execute the flow [49](#)

F

flow description language [34](#)
flow.tcl file [35, 48](#)
FlowTracer Advanced Tutorials [45](#)
FlowTracer Beginner's Tutorial [3](#)
forget nodes and sets from the graph [28](#)

G

grid view [23](#)
GUI job views [21](#)

H

horizontal graph view [23](#)

M

make changes to a file and run it [25](#)

N

navigate the graph [25](#), [26](#)

R

register one job from the command line [9](#)

remove depended input file [17](#)

remove older sets [34](#)

repeat the tutorial without the GUI [32](#)

rerun from the CLI [30](#)

restore the shell prompt [6](#)

run a flow from the command line [38](#)

run the flow interactively [37](#)

run the jobs [25](#)

S

set command line environment [4](#)

start the GUI console [7](#)

stat view [24](#)

stop the project [44](#)

U

use the set browser [8](#)

V

vertical graph view [22](#)

view graph subsets [24](#)

W

write flows [48](#)