



ALTAIR

ONLY FORWARD

Altair Accelerator Software 2024.1.0

Tutorials

Contents

Art of Flows Example Guide	5
Example 1.....	6
Example 2.....	9
Example 2 with Scripts.....	9
Altair Accelerator User Tutorials	17
Use Accelerator Help.....	18
Enable CLI Access on UNIX.....	20
Troubleshooting the UNIX Setup.....	22
Enable CLI Access on Windows.....	23
Verify Context Is Working.....	24
Run Basic Jobs.....	26
Get Summary Information.....	28
Run Jobs with Various Options.....	31
Submit Multiple Jobs at Once: -f <file>.....	31
Use Environments: -e <env>.....	31
Use of Resources: -r <res1 res2 ...>.....	32
Wait for Jobs.....	32
Specify Name of logfile: -l <logfile>.....	33
Job Control.....	34
Rerun Jobs.....	36
Get Detailed Information about a Job.....	37
Monitor Jobs, Taskers and Resources.....	39
Invoke the GUI.....	41
Use the Web Browser.....	43
Troubleshooting.....	44
Altair Accelerator Administrator Tutorials	45
Start a Test Queue.....	46
Start/Stop Accelerator.....	49
Browser-based Setup.....	51
Configure Policy - FairShare and Other Parameters.....	52
Server Configuration Parameters.....	53
Advanced Policy Configuration.....	54
Configure Resources.....	56
Configure Security.....	58
Configure Taskers.....	60
Configure an Environment.....	65

Logical Names (Equivalences).....	70
Resource Management.....	73
Altair Monitor-Basic Setup.....	73
Configure and Manage Monitor-basic.....	74
vtk_flexlm_monitor Procedures.....	75
Resource Throttling.....	75
Upgrade Accelerator.....	76
Use Altair Accelerator's REST API to Submit and List Jobs.....	79
Key REST Concepts.....	80
Resource Path.....	80
Get Ready for REST.....	82
Example Application: REST 101.....	83
Launch Jobs with Non-default Options.....	87
The vov_rest_v3.py Python Library Module.....	88
REST Request Detailed Documentation.....	90
Issuing REST Requests from the Swagger Web UI.....	91
Advanced REST Usage.....	93
nc_info.py.....	94
JWT Access Tokens.....	95
HTTPS Security Considerations.....	95
Connection Keep-Alive.....	95
JWT Token Allocation.....	96
Case Study.....	98
EDA Automation Tutorial.....	101
EDA Demo Part 1: Run the Demo.....	102
Setup.....	102
Start the Project.....	102
Customize the Project.....	103
Check Out the Data.....	103
Start the Browser Interface.....	104
EDA Demo Part 2: Dissect the Demo.....	110
The Chip Structure File and cdt Script.....	110
The Capsules.....	110
The Flow Description BlockFlow.tcl.....	111
The CGI script edademo.cgi.....	113
FlowTracer Beginner's Tutorials.....	120
Create a FlowTracer Project.....	121
Set Command Line Environment.....	121

Create a Project.....	122
Enable a Shell.....	123
Restore the Shell Prompt.....	123
Check Project Information.....	123
Start the GUI Console.....	124
Use the Set Browser.....	125
Add a Job Interactively.....	125
GUI Job Views.....	138
Navigate the Graph.....	142
Command Line Interface.....	146
Flow Description Language.....	151
Remove Older Sets.....	151
The Flow.tcl File.....	152
Build the Flow.....	153
Run the Flow Interactively.....	154
Run a Flow from the Command Line.....	155
Batch Process to Define and Run a Flow.....	156
Create a Complex Flow.....	157
EDA Flows.....	159
Stop the Project.....	161
FlowTracer Advanced Tutorials.....	162
Create Efficient VOV Scripts.....	163
Write Flows.....	165
The Flow.tcl file.....	165
Build the Flow.....	165
Execute the Flow.....	166
Build a More Complex Flow.....	166
EDA Flows.....	166
Generate Custom HTML Reports Using CGI Tutorial.....	168
Job Reports: List.....	170
Job Reports: Table.....	171
Legal Notices.....	172
Intellectual Property Rights Notice.....	173
Technical Support.....	177
Index.....	179

This chapter covers the following:

- [Example 1](#) (p. 6)
- [Example 2](#) (p. 9)

The examples described in *The Art Of Flows* book are available in the `$VOVDIR/training/art_of_flows` directory. *The Art Of Flows* is available in the documentation bookshelf in PDF form. This section describes how to run the examples.

To begin, please point to the new environments directory and switch to the environment EDA1:

```
% setenv VOV_ENV_DIR $VOVDIR/training/art_of_flows/environments
% ves EDA1
```

Example 1

```
% mkdir aof_ex1
% cd aof_ex1
% date > Block.v
```

1. To run Example 1, create a directory and create a file called `Block.v`.

```
% mkdir aof_ex1
% cd aof_ex1
% date > Block.v
```

2. Start a FlowTracer project and start a GUI:

```
% vovproject create art_of_flows
% vovproject enable art_of_flows
% vovconsole -view graph -set All:nodes &
```

Example 1 with Scripts

The first script is "naked", the second has more frills.

```
% $VOVDIR/training/art_of_flows/example1/script1_1.csh Block.v Block.vg
% $VOVDIR/training/art_of_flows/example1/script1_2.csh Block.v Block.vg
```

Example 1 with make

The first makefile is simple, the second tries to augment the information about how the job is executed by printing additional information on stdout.

```
% make -f $VOVDIR/training/art_of_flows/example1/Makefile1_1
% make -f $VOVDIR/training/art_of_flows/example1/Makefile1_2
```

If you have Accelerator, you can also try a makefile that has hard-coded links to a specific scheduler:

```
% make -f $VOVDIR/training/art_of_flows/example1/Makefile1_3
```

Example 1 with FlowTracer

The two flow descriptions yield exactly the same result.

```
% vovbuild -f $VOVDIR/training/art_of_flows/example1/Flow1_1.tcl
% vovbuild -f $VOVDIR/training/art_of_flows/example1/Flow1_2.tcl
```

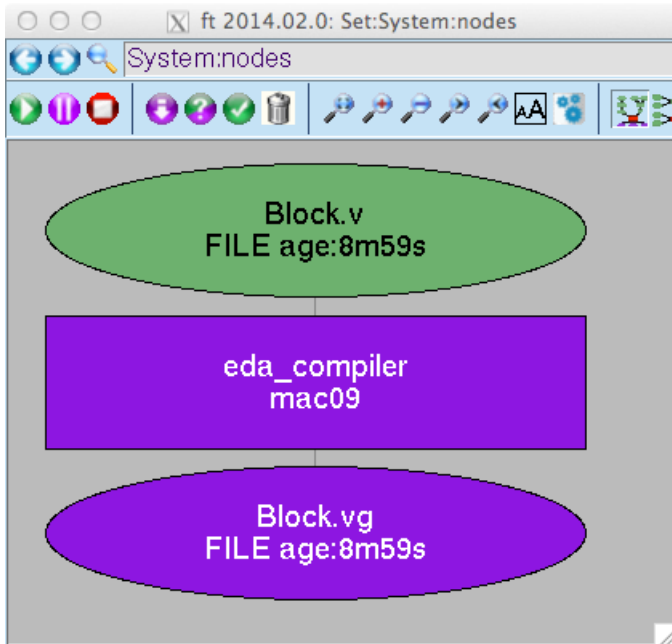


Figure 1: Flow is Built

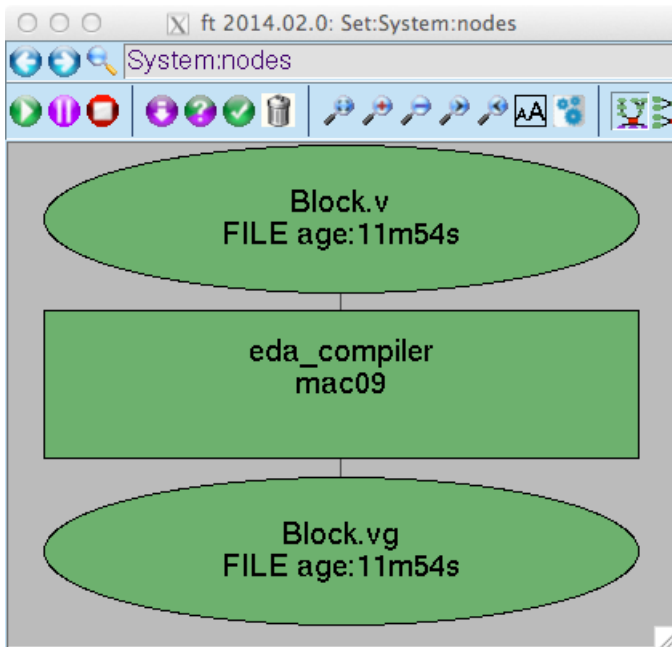


Figure 2: Flow is done

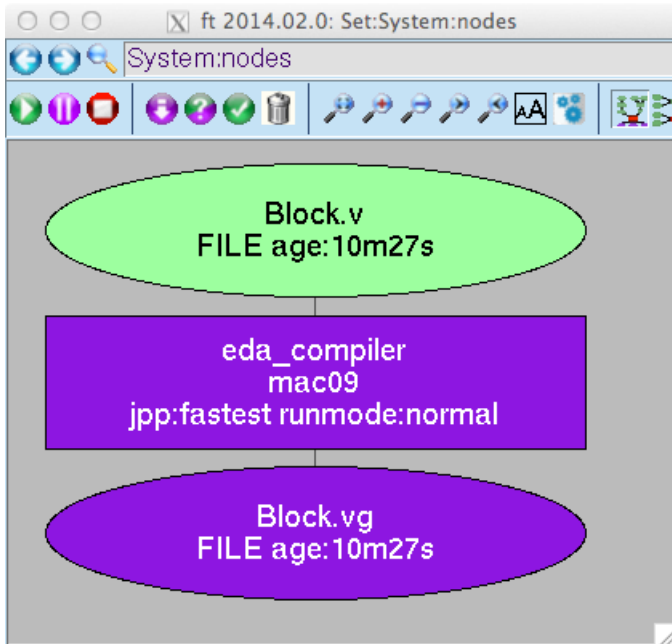


Figure 3: Flow has been changed

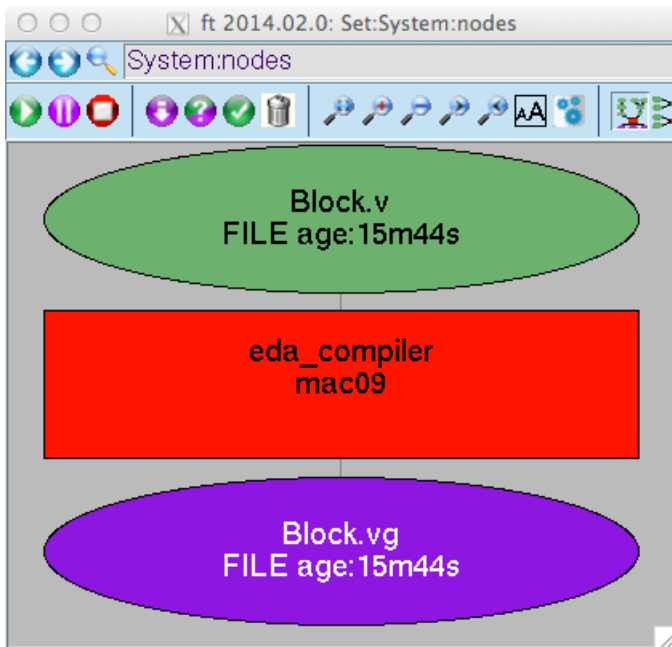


Figure 4: Flow is failing

Example 2

To run Example 2, create a directory and initialize it with the script `create_example_2`

```
% mkdir ex2run  
% cd ex2run  
% create_example_2
```

Example 2 with Scripts

Example 2 with Scripts

The first script is trivial, the second has a bit of error checking:

```
% $VOVDIR/training/art_of_flows/example2/script2_1.csh  
% $VOVDIR/training/art_of_flows/example2/script2_2.csh
```

Example 2 with make

This is an example of a recursive makefile system:

```
% make all
```

Example 2 with FlowTracer

The `Flow.tcl` file creates a multi-directory flow that is easy to manage.

```
% vovbuild  
% vsr -all
```

To create a Makefile or a script from the flow, you can use `vovexport`:

```
% vovexport -make  
% vovexport -csh
```

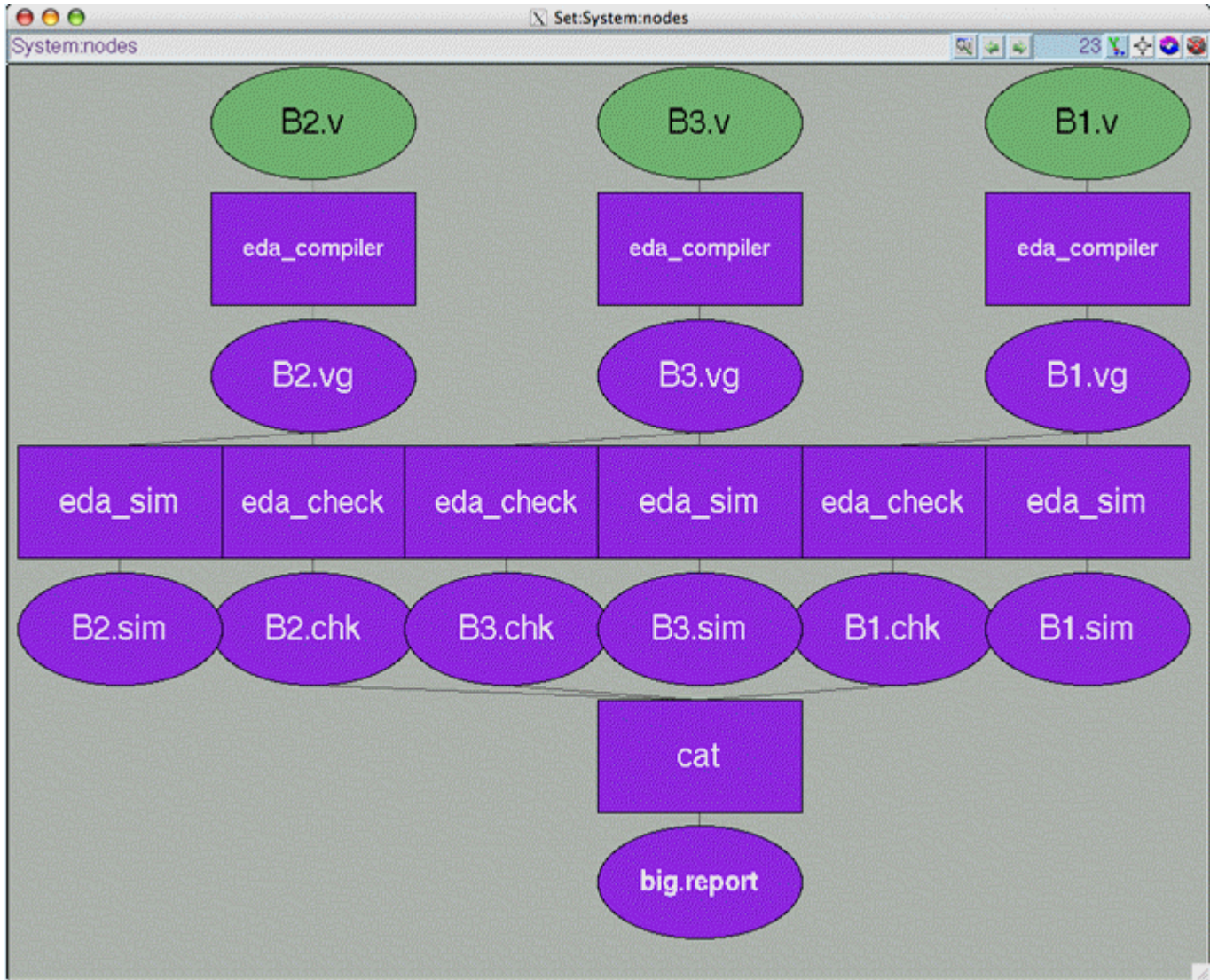


Figure 5: Flow is built

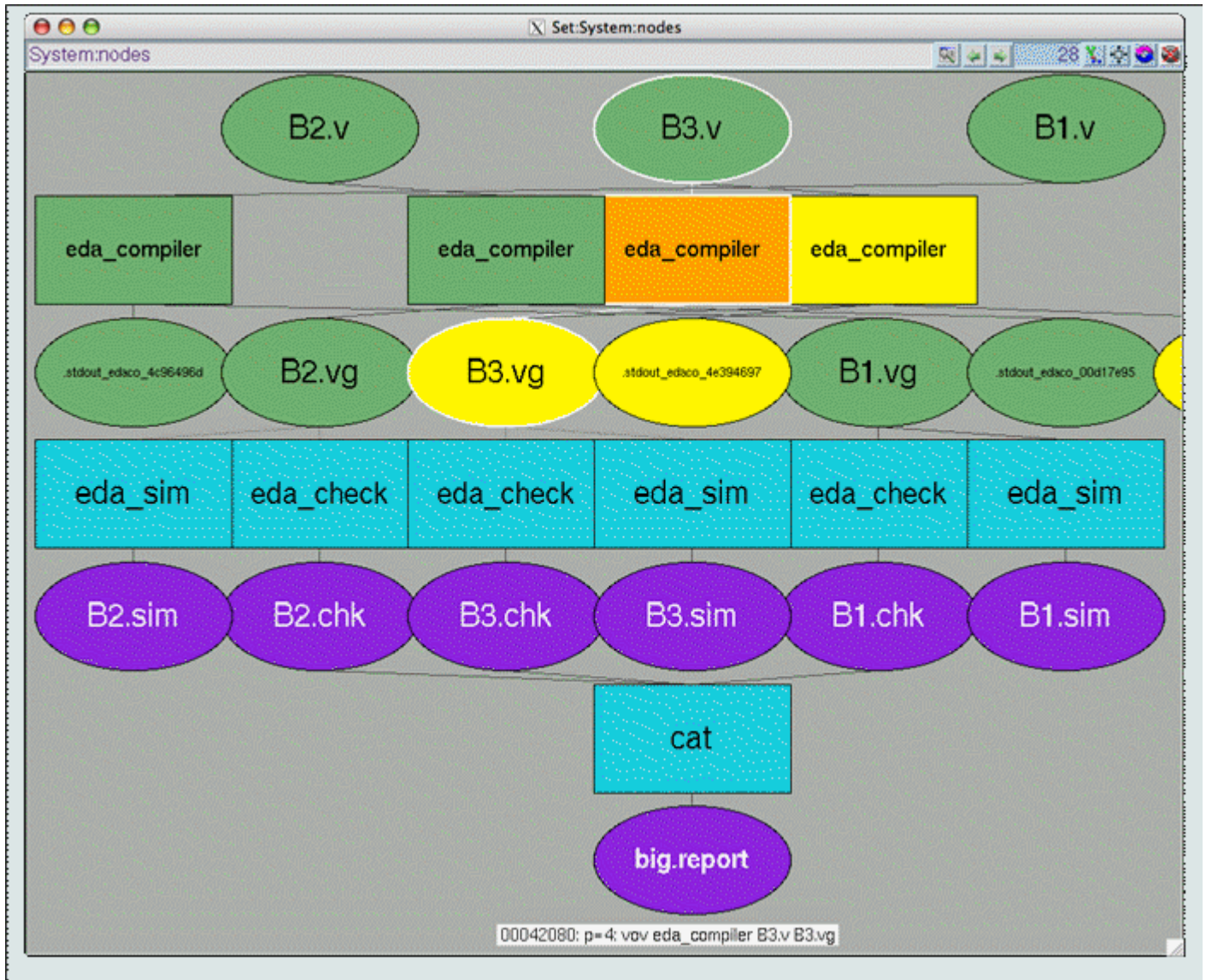


Figure 6: Flow is running

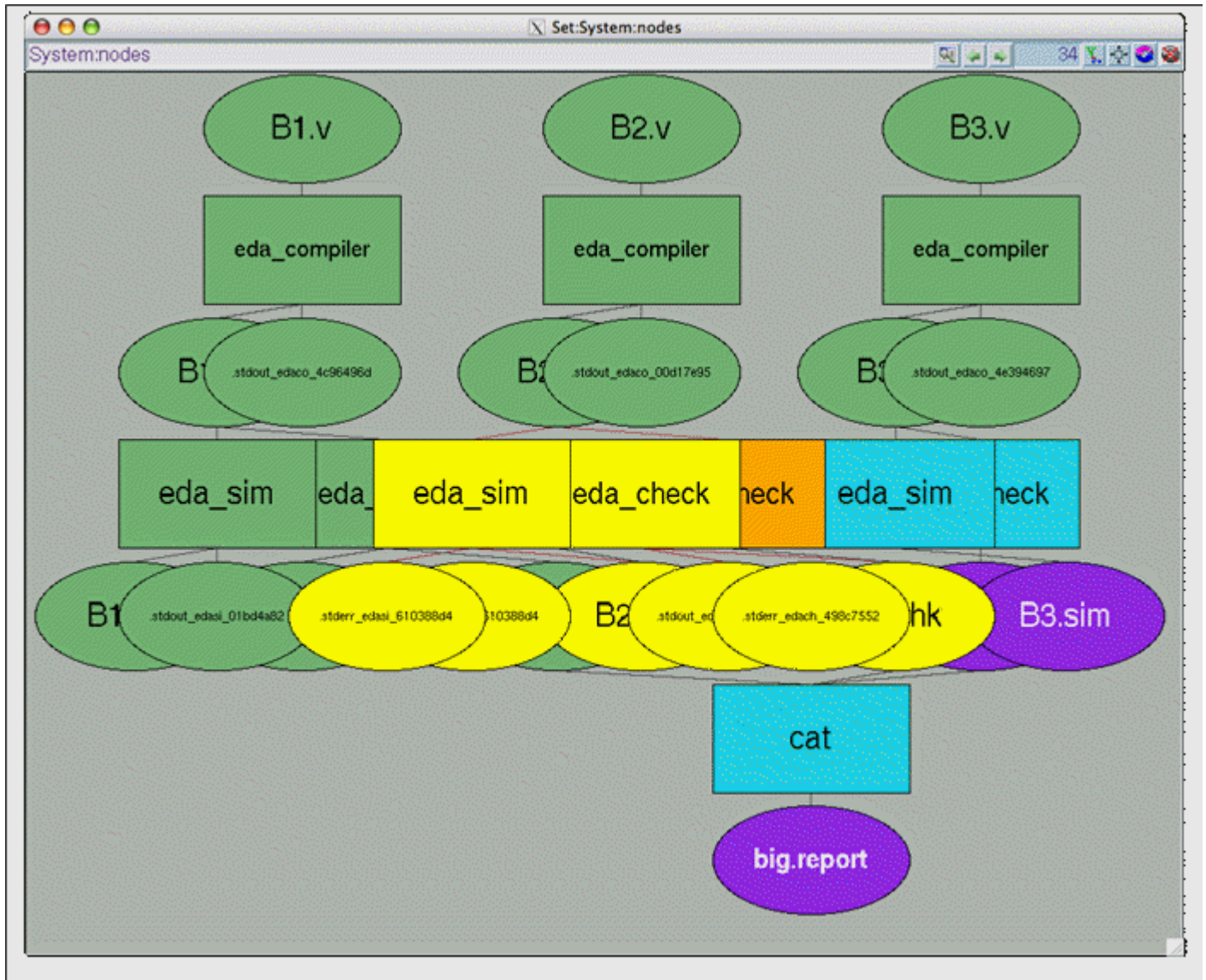


Figure 7:

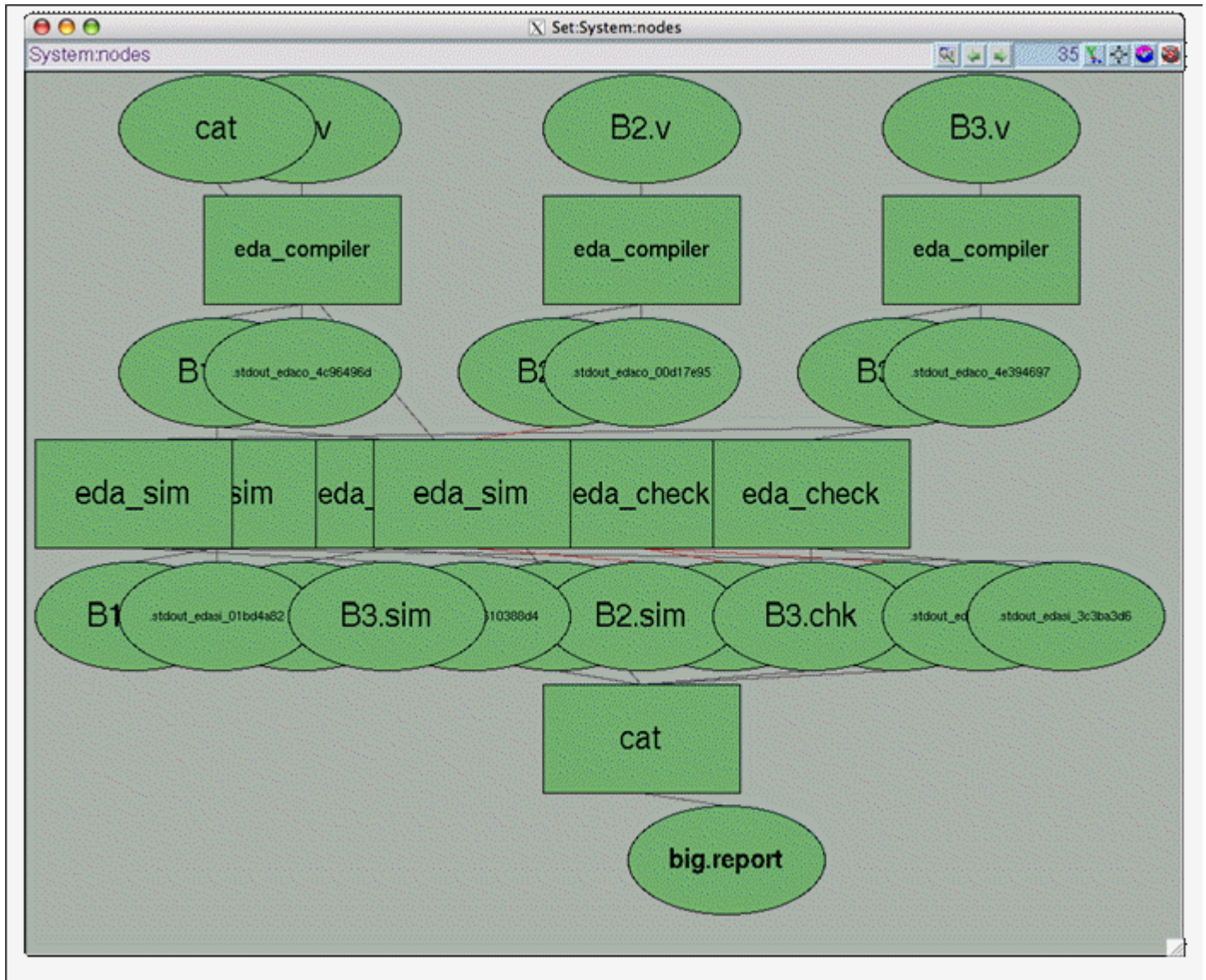


Figure 8: Flow is done

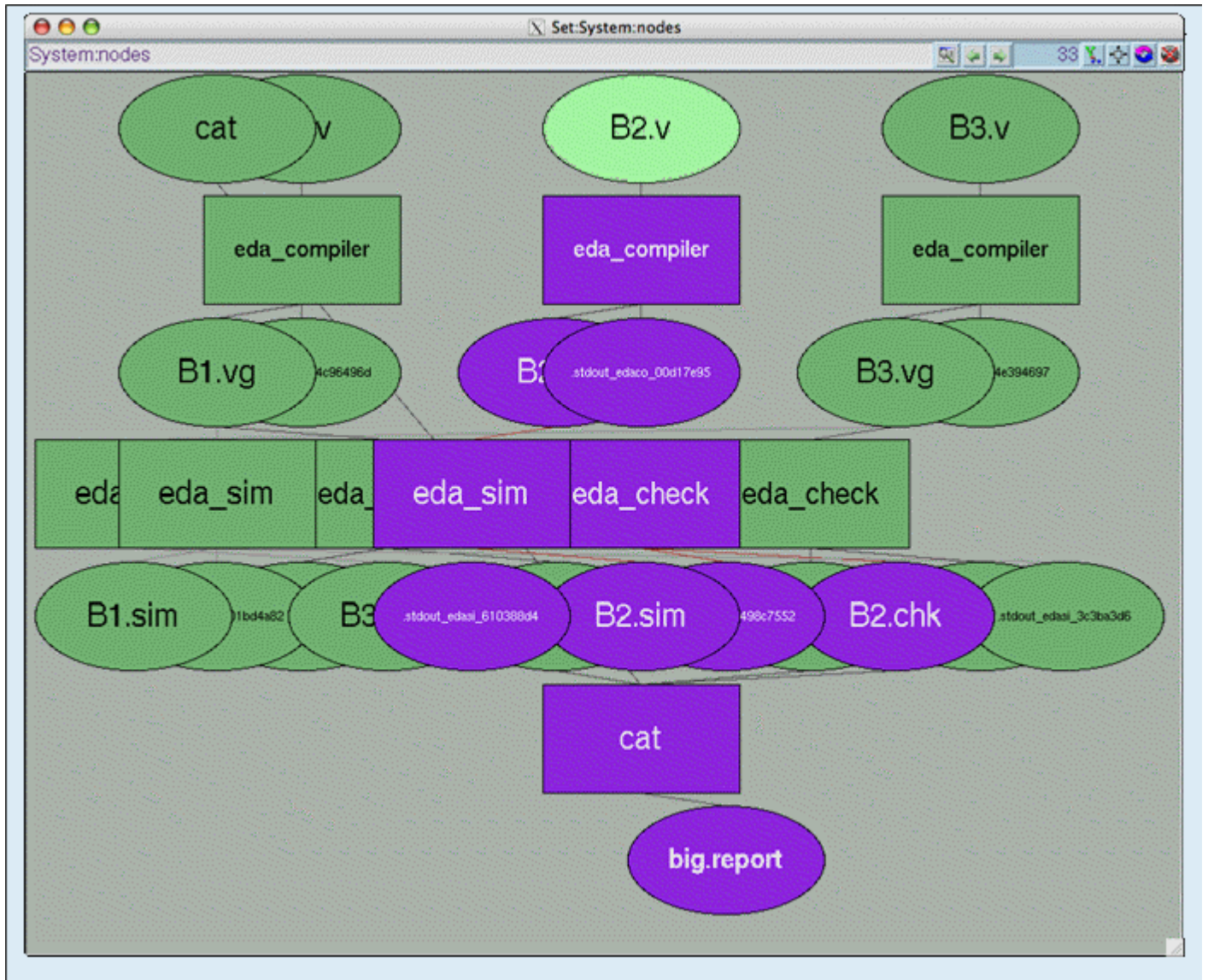


Figure 9: Flow is changed

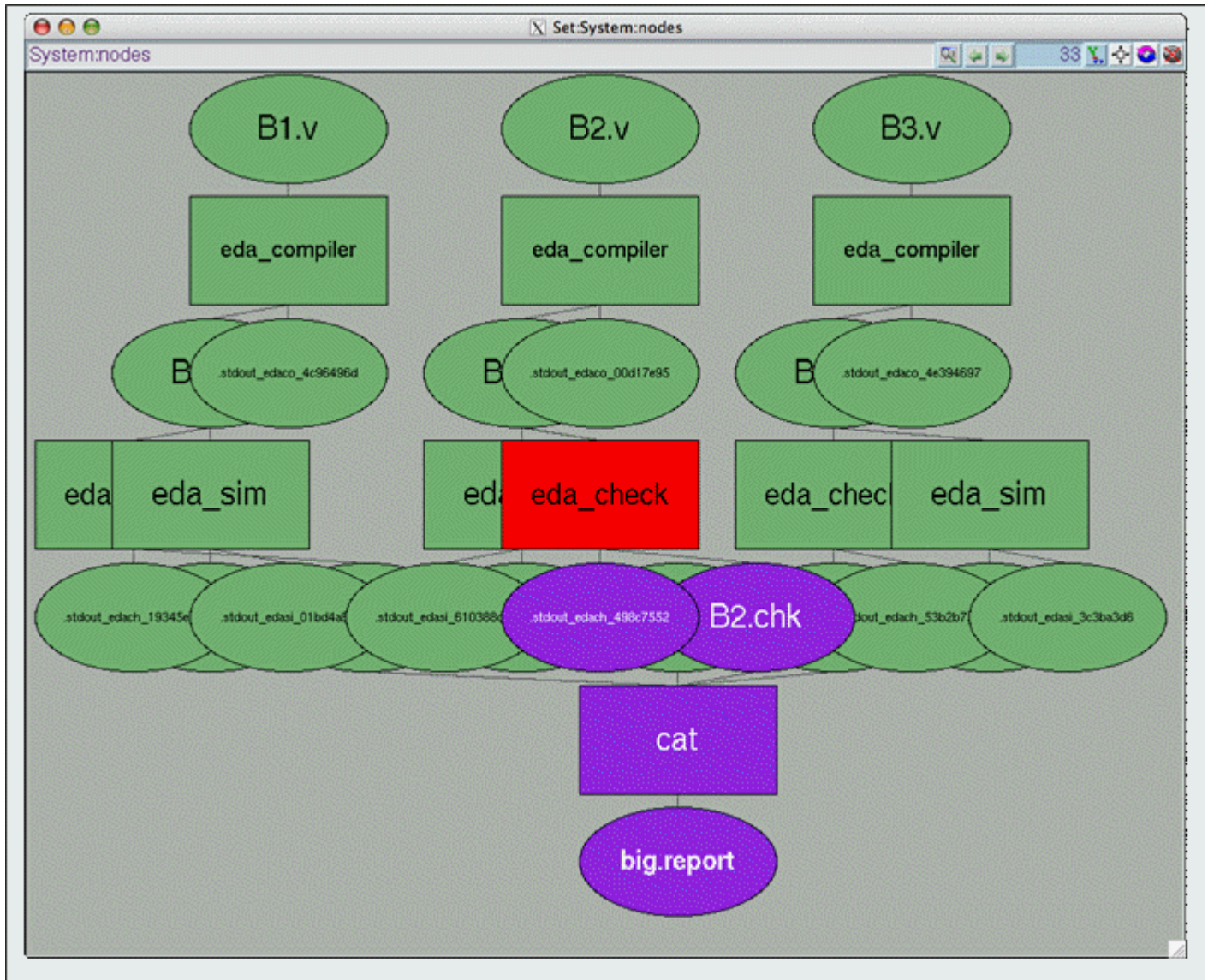


Figure 10: Flow is failing

Example 2 with Makefile and FlowTracer

To convert the Makefile into a flow, you can use vovmake. The behavior of vovmake is controlled by vovmake.config.tcl. In this case the configuration file simply requests:

- The skipping of the targets 'all' and 'run'
- The use of the wrapper vw for most other targets
- The use of the environment EDA1

```
% vovforget -allnodes ; if necessary, to cleanup the old flow.  
% vovmake
```

You can then create a new makefile using vovexport

```
% vovexport -make
```

```
% make -f Makefile.vov clean  
% make -f Makefile.vov all
```


In the following User Tutorials, you will experiment with Accelerator on most topics that a user would be most interested in, including submitting jobs, tracking job information, analyzing and solving common problems, etc.

This chapter covers the following:

- [Use Accelerator Help](#) (p. 18)
- [Enable CLI Access on UNIX](#) (p. 20)
- [Enable CLI Access on Windows](#) (p. 23)
- [Run Basic Jobs](#) (p. 26)
- [Get Summary Information](#) (p. 28)
- [Run Jobs with Various Options](#) (p. 31)
- [Job Control](#) (p. 34)
- [Rerun Jobs](#) (p. 36)
- [Get Detailed Information about a Job](#) (p. 37)
- [Monitor Jobs, Taskers and Resources](#) (p. 39)
- [Invoke the GUI](#) (p. 41)
- [Use the Web Browser](#) (p. 43)
- [Troubleshooting](#) (p. 44)

Also in This Tutorial

Use Accelerator Help

Accelerator documentation is available in HTML and PDF format.

Access the Help when Accelerator is Running

When Accelerator is running, it displays the documentation through its browser interface. To access it from browser, you need to know which host and port Accelerator is running on. Ask your administrator, or find the URL for Accelerator with the following command:

```
% Accelerator cmd vovbrowser
http://comet:6271/project
```

In the example below, assume Accelerator is running on host comet, port 6271. The URL for Accelerator is:

```
http://comet:6271
```

To get the entire suite of Altair Accelerator documents, including FlowTracer™, Accelerator™, Monitor™ and the VOV subsystem, use the following URL:

```
http://comet:6271/doc/html/bookshelf/index.htm
```

Access the Help when Accelerator is not Running

All the documentation files are in the Altair Accelerator install directory, so you can access them even if vovserver is not running. To do this, open `/installation_directory/common/doc/html/bookshelf/index.htm` in your browser.

 **Tip:** Bookmark the above URL for future reference.

Access the Help PDF Files

Altair Accelerator also provides PDF files for each of the guides. All the PDF files are in the directory `/installation_directory/common/doc/pdf`

Access the Help via the Command Line

The main commands of Accelerator are `nc` and `ncmgr`, with some subcommands and options. You can get usage help, descriptions and examples of the commands by running the command without any options, or with the `-h` option. For example,

```
% nc info -h
nc:
nc:  NC INFO:
nc:  Get information about a specific job or list of jobs.
nc:  USAGE:
nc:  % nc info <jobId> [options]...
nc:  -h          -- Show this message
nc:  -l          -- Show the log file
nc:
```

Access the Help via the vovshow Command

Another source of live information is using the command `vovshow`. The following options are often useful:

- vovshow -env RX** Displays the environment variables that match the regular expression RX provided.
- vovshow -fields** Shows the fields known to the version of VOV in use.
- vovshow -failcodes** Shows the table of known failure codes.

For example, to find a variable that controls the name of the stdout/stderr files, without knowing the exact name of that variable, the following command can be used:

```
% vovshow -env STD
VOV_STDOUT_SPEC          Control the names of file used to save stdout and
                          stderr. The value is computed by substituting
                          the substrings @OUT@ and @UNIQUE@ and @ID@.
                          Examples: % setenv VOV_STDOUT_SPEC
                          .std@OUT@.@UNIQUE@ % setenv VOV_STDOUT_SPEC
                          .std@OUT@.@ID@
```

The output provides a description of all the variables used by the FlowTracer system that include the substring "STD". In this example, the output result VOV_STDOUT_SPEC.

Enable CLI Access on UNIX

This section explains how to set up a UNIX user's shell environment to have a proper context for the user to run installed Altair Accelerator programs from the command line. The programs that are run from the command line are called the CLI commands.

When a UNIX user logs into the system, a login script is executed which sets up their working environment. The default shell for the user determines what login script will run. A csh shell uses a `.cshrc` login script. A bash shell uses a `.profile` login script. The best way to set up a user's shell environment to run Altair Accelerator programs is to change the login script to properly set the user's working environment.

Altair Accelerator products require that the `PATH` environment variable be set to include the directories in the Altair Accelerator release which hold programs that will be run from the command line. Also, there are two environment variables that need to be set so that when an Altair Accelerator program is run, it will know where the release is installed and know what type machine it is running on. These two environment variables are `VOVDIR` and `VOVARCH`.

The Altair Accelerator product installation provides a helper file that can be sourced in order to set the needed environment variables to values that are appropriate for the local situation.

The intended use of the helper file is to have it sourced from a user's shell login script so that the user gets a proper environment without doing anything extra.

There is one helper file for each of the shell types that exist on UNIX. One helper script file is in the csh syntax and the other is in the shell syntax.

- `vovrc.csh`
- `vovrc.sh`

You can change each user's shell login script to source the file that is appropriate for the particular shell that they use.

Shell	Instructions
C-shell, tcsh	Add the following line to your <code>.cshrc</code> csh login script file: <pre>source /<install_path>/<version>/<platform>/etc/ vovrc.csh</pre>
sh, ksh, bash, zsh	Add the following line to your <code>.profile</code> shell login script file: <pre>./<install_path>/<version>/<platform>/etc/ vovrc.sh</pre>

Verify Access to Altair Accelerator Products

After making the changes to source the helper files, and then logging in, you can check that needed environment variables are set properly by looking at environment variables to note that the `PATH` value contains a reference to the folders in the release

which hold programs, and to note that VOVDIR and VOVARCH are set. VOVDIR should contain the path to the installation. VOVARCH should contain the name that matches the machine type it is running on.

1. At the command prompt, enter the following:

```
% echo $PATH
% echo $VOVDIR
% echo $VOVARCH
```

Beyond just looking, you can try running a Altair Accelerator product program as a positive test that the context is set up properly. One program that is simple to run and is available with all Altair Accelerator products is `vovarch`. This program reports on the machine type on which it is running. A side effect is that it tests that needed environment variables are set properly. If the environment is valid, the program will run and report the correct machine type. If the environment is not valid, the program will not work.

2. From the shell prompt, run the command `vovarch`.

```
% vovarch
```

You should get a response that is similar to one of the following, "linux64", or "macosx". This indicates the program was found, it ran, and it produced an expected output that matches your UNIX environment.

You have successfully set up the environment and verified it is correct.

3. If the response is `Command not found`, then the working environment does not have a VOVARCH setting for the programs in the Altair Accelerator products install area. If this is the case, review the steps to make sure the helper file from Altair Accelerator is being sourced correctly.

```
% vovarch
linux64
... or ...
macosx
```

or

```
% vovarch
vovarch: Command not found.
```

Enable Altair Accelerator for Non-Interactive Shells

It is possible to have the shell login script build a different working environment for interactive and non-interactive shells. The non-interactive environment is used when you run a batch script.

A common way this is done is to do an early exit from the login script file for non-interactive shells. For CSH, this can be done by testing for the existence of the shell variable "prompt".

Early exit from non-interactive shell login CSH script:

```
# This is a fragment of .cshrc.
:
# Batch scripts can skip doing actions needed by interactive scripts.
if ( ! $?prompt ) then exit
:
```

```
# Below are actions needed by interactive scripts.  
:
```

If this exit happens early in the script, before sourcing the Altair Accelerator helper file, then the environment variables will not be set for the non-interactive shell.

A batch script needs access to Altair Accelerator products. This means that the non-interactive shell needs to have the needed environment variables set.

You should place the code that sources the helper file from Altair Accelerator early in the shell login script, before any logic causes the script to differ between interactive and batch shells.

Source helper file before exit in login CSH script:

```
# This is a fragment of .cshrc.  
:  
# Altair Accelerator env vars are needed by batch scripts.  
source /<install_path>/<version>/<platform>/vovrc.csh  
:  
# Batch scripts can skip doing actions needed by interactive scripts.  
if ( ! $?prompt ) then exit  
:  
# Below are actions needed by interactive scripts.  
:
```

Enable the Shell to Communicate With a Running Product Server

If you need to issue commands that will communicate to a running product instance, the instance will need to be "enabled", which involves setting certain environment variables that point the commands to the location of the running vovserver.

```
% vovproject enable instance-name
```

Some common instance names are "licmon" (for Monitor), "vnc" (for Accelerator), and "wx" (for Accelerator Plus). Instance names can be anything though since they are user-definable upon first start of each product.

Troubleshooting the UNIX Setup

An earlier section of this manual explained the importance of editing the `.cshrc` file so that batch shells would have the proper environment for running Altair Accelerator products.

The following is a command line to verify that the `.cshrc` file is set properly for batch shells. This runs the `vovarch` command within a batch context.

```
% csh -c vovarch
```


If this fails, the `.cshrc` file is not edited properly to enable access for batch shells. Review the details of the earlier topic on editing the `.cshrc` to enable access to batch shells.

Enable CLI Access on Windows

This section explains how to set up a Windows user's command prompt environment to have the proper context for the user to run installed Altair Accelerator programs from the command line. The programs that are run from the command line are called the CLI commands.

Altair Accelerator products require that the PATH environment variable be set to include the directories in the Altair Accelerator release which hold programs that will be run from the command line. Also, there are two environment variables that need to be set so that when an Altair Accelerator program is run, it will know where the release is installed and know what type of machine it is running on. These two environment variables are VOVDIR and VOVARCH.

There are two methods for setting the correct environment. Both involve running the same context-setting bat script. This context-setting bat script establishes the correct environment variables for the local situation, reflecting where Altair Accelerator products are installed.

 **Note:** This operation to set the environment is not required to use every Altair Accelerator feature on Windows. This operation is only needed to enable using the CLI commands from the command prompt.

Method 1: Use Windows Explorer to Set Command Line Environment

1. Using Windows Explorer, navigate to the Altair Accelerator installation directory.
2. Enter the `win64/startup` folder and double-click the `vovcmd.bat` script to run it.
This will open a command prompt with the proper environment settings for the Altair Accelerator and scripts to work.

When `vovcmd.bat` runs, it will execute the `win64/bat/vovinit.bat` script as part of what it does. The following section covering Method 2 explains what `win64/bat/vovinit.bat` does when it runs. It does the same thing when run by either method.

Method 2: Using Windows Command Prompt to Set Command Line Environment

1. In a command prompt window, navigate to the Altair Accelerator installation directory using the `cd` command.
2. Change directory to the `win64/bat` folder with `cd` and run the `vovinit.bat` script.
This will establish the needed environment for the open command prompt.

When `win64/bat/vovinit.bat` runs, it figures out needed environment variables and sets them, based upon where it is located. In particular, it sets VOVDIR to be the path to where Altair Accelerator is installed. It then executes another initialization script, `$VOVDIR/win64/local/vovinit.bat`, if it exists.

This is an initialization script that you can create and modify to perform site specific activities customized for your local configuration and usage. You can add commands to the `local/vovinit.bat` file that you want to run whenever a user starts up a command prompt.

After running both `vovinit.bat` scripts, the context of the command prompt has the correct environment needed so that CLI commands will work correctly with the installed Altair Accelerator.

Customize Actions Needed to Enable Access to Altair Accelerator Products on Windows

You can add commands to the `win64/local/vovinit.bat` file that perform specific operations that enable your Windows users to access Altair Accelerator programs properly, or to set up things on the machine to follow a standard convention.

You could add operations that perform actions such as these, and others:

- Mounting network drives
- Setting environment variables for local needs
- Establishing time synchronization

Example of a custom `$VOVDIR/win64/local/vovinit.bat`

```
rem -- Mount network drives:
rem -- In this example we mount the Altair Accelerator installation on drive v:
if not exist v:\nul net use v: \\somehost\altair

rem -- Set locally useful environment variables.
set VNCSWD=v:\vnc
set DLOG=d:\dailylog

rem -- Set Windows time from server on local network
rem -- Put this last; it may fail if lacking time set privilege
net time \\timehost /set /y
```

Verify Context Is Working

When you have a command prompt open and expect that it has a context for accessing Altair Accelerator programs, check that the environment is set by looking at environment variables to note that the `PATH` value contains a reference to the folders in the release which hold programs, and to note that `VOVDIR` and `VOVARCH` are set. `VOVDIR` should contain the path to the installation. `VOVARCH` should contain the name that matches the machine it is running on.

Beyond just looking, you can try running a program as a positive test that the context is set up properly. One program that is simple to run and is available with all Altair Accelerator products is `vovarch`. This program reports on the machine type on which it is running. A side effect is that it tests that needed environment variables are set properly. If the environment is valid, the program will run and report the correct machine type. If the environment is not valid, the program will not work.

Run `vovarch` to verify the environment is set ok:

```
c:\ > vovarch
win64
```




Note: The output will always show **win64** when running on any of the Microsoft Windows operating systems. This result is expected. It reports that we are on a generic "windows" architecture and indicates that the context is working.

If you are not able to verify that the context is valid, check the details within the <installation_directory>/<version>/<platform>/bat/vovinit.bat file.

Enable the Command Prompt to Communicate With a Running Product Server

If you need to issue commands that will communicate to a running product instance, the instance will need to be "enabled", which involves setting certain environment variables that point the commands to the location of the running vovserver.

```
c:\ > vovproject enable instance-name
```

Some common instance names are "licmon" (for Monitor), "vnc" (for Accelerator), and "wx" (for Accelerator Plus). Instance names can be anything though since they are user-definable upon first start of each product.

Run Basic Jobs

Submitting jobs to Accelerator is quite simple: just use `nc run` followed by the command you would use without Accelerator.

Run a Sleep Job


```
% cd # Go to your home directory
% nc run sleep 10
Resources= linux
Env = D(VOV_ENV_SOURCE=vnc_logs/envdixin36362.env)
Command = vw vwrap sleep 10
Logfile = vnc_logs/20021230/103409.13784
JobId = 04194422
nc: message: Scheduled jobs: 1 Total estimated time: 0s
```

Run Sleep Job with `-r` Option to Override Default Resources

By default, `nc run` takes the architecture of the machine from which the job is submitted as the job's resource. This can be overridden with `-r` option when needed.

This job requires no resource.

```
% nc run -r "" -- sleep 10
Resources=
Env = D(VOV_ENV_SOURCE=vnc_logs/envdixin36362.env)
Command = vw vwrap sleep 10
Logfile = vnc_logs/20021230/103653.13794
JobId = 04194459
nc: message: Scheduled jobs: 1 Total estimated time: 0s
```

 **Note:** Since the `-r` option accepts multiple arguments, you need to terminate the resource list explicitly with another option, or `--` if no other options are needed.

This job requires `spice_license`.

```
% nc run -r spice_license -- sleep 10
Resources= spice_license
Env = D(VOV_ENV_SOURCE=vnc_logs/envdixin36362.env)
Command = vw vwrap sleep 10
Logfile = vnc_logs/20021230/103948.13803
JobId = 04194497
nc: message: Scheduled jobs: 1 Total estimated time: 0s
```

Run Job with `-e` Option to Override Default Environment

By default, `nc run` takes a snapshot of the environment as the job's environment and uses this to run your job. You can override this with `-e` option to use a named environment.

```
% nc run -e BASE sleep 10
Resources= linux
Env = BASE
Command = vw vwrap sleep 10
Logfile = vnc_logs/20021230/122737.14946
JobId = 04195924
```

```
nc: message: Scheduled jobs: 1          Total estimated time: 0s
```

Use the command `vel` to list all the available environments:

```
% vel
vel: message: Environment directories:
1 /remote/release/VOV/7.0u3/linux64/local/environments
1 . tcl BASE          UNIX utilities, X windows, and VOV.
1 . tcl D             Define variables: Usage: ves "+D(VAR1=value1,...)"
1 * tcl DEFAULT      Just a name for whatever you already have.
1 . tcl HSIM         Fake Nassda hsim env for Virage testing
1 . csh SYNOPSIS     Synopsys tools
```

Scripts that implement named environments may be written in `csh`, `sh`, or `Tcl` syntax. Cross-platform environments between Windows and UNIX/Linux must be written in `Tcl`. You can use named environments from the command line using the command `ves`.

Get Summary Information

nc summary, nc list and nc info *jobId* will be the three commands that you use frequently to get information about the jobs managed by Accelerator.

Get Summary of All My Jobs: nc summary

```
% nc summary
NC Summary For User dextrin
TOTAL JOBS      11      Duration: 23s
  Done          1
  Idle          0
  Queued        5
  Running       2
  Failed        3

  JOBS  GROUP  TOOL      WAITING FOR...
    1   alpha  sleep     'aa'
    3   alpha  sleep     'linux'
```

In the above example, the command displays summary information for user dextrin. There are a total of 11 jobs, 1 of which is Done, 2 Running, 3 Failed, etc. In the bottom part, it shows the summary for jobs that are queued, that is, there is 1 sleep job queued because it is waiting for resource 'aa', and there are 3 sleep jobs waiting for resource 'linux'.

To get an idea of what is going on in the whole Accelerator system, the following shows all the jobs and what the jobqueue buckets are waiting for. Note that Accelerator subcommands may be abbreviated.

```
% nc sum -a -b
```

Get a List of My Jobs: nc list

This command, without any option, displays the last 20 of your jobs in the format of "jobId status command".

```
% nc list
04193437 Done      sleep 1
04193911 Done      sleep 5
04193913 Failed    sleep 10
04193917 Failed    sleep 20
04193919 Idle      sleep 60
04193937 Running   sleep 10
04193943 Queued    sleep 56
```

Status Meaning

In Accelerator, each job is assigned an "Accelerator Computing Status" which is defined as follows:

Status	Color *	Explanation
Queued	Cyan	The job is scheduled to be executed.
Running	Orange	The job is currently executing
Done	Green	The job ran successfully

Status	Color *	Explanation
Failed	Red	The job ran and failed.
Idle	BlueViolet	The job needs to be run, but it is not scheduled.

*(Colors may look different on some systems)

Some Options about nc list

The `nc list` command has several useful options. They include:

Get detailed usage of this command:

```
% nc list -h
```

List all jobs, including others' jobs:

```
% nc list -a
04193437 Done    alpha    dextrin  rhino    sleep 1
04193898 Done    users   integ    rhino    sleep 2
04193939 Done    alpha    dextrin  rhino    sleep 15
04193941 Done    alpha    dextrin  rhino    sleep 20
04193943 Done    alpha    dextrin  rhino    sleep 60
```

Control output format, show job Id the (first) tool of job only:

```
% nc list -O "@ID@ @TOOL@"
04193437 sleep
04193898 sleep
04193939 sleep
04193941 sleep
04193943 sleep
```

In the above example, we use "fields", i.e., ID and TOOL, surrounded by two "@" signs, to format strings.

Get Detailed Information About a Job: nc info jobId

Without options, this command displays the basic information about the job, like user, group, directory, command, environment, queue time, etc.

```
% nc info 04193937
Id,User,Group    04193937,dextrin,alpha
Environment      D(VOV_ENV_SOURCE=vnc_logs/envdextrin36362.env)
Directory        ${HOMES}/dextrin
Command          sleep 10
Status           Done
Host             rhino
QueueTime        1s
Duration         10s
Age              20m37s
AutoForget       1
```

With option -l, this command shows the contents of the log file.

```
% nc info -l 04193937
Log file is: '${HOMES}/dexin/vnc_logs/20021230/095337.13512.3'
vwrap: message: Start date: Mon Dec 30 09:53:38 PST 2002
vwrap: message: On host: rhino
vwrap: message: Sourcing environment vnc_logs/envdexin36362.env
vwrap: message: Running: 'sleep 10'
vwrap: message: Exit status: 0
vwrap: message: End    date : Mon Dec 30 09:53:48 PST 2002
```

Run Jobs with Various Options

In this section, we will exercise some options in `nc run` command.

To get the detailed usage, use the following command:

```
% nc run -h
```

Submit Multiple Jobs at Once: `-f <file>`

1. Prepare a file with one command on each line. Empty lines are ignored and lines that begin with `#` are considered comments.

```
# Example of file used to submit multiple jobs at once.  
sleep 10  
sleep 11  
sleep 12  
sleep 13
```

2. Use the option `-f` to specify the command file, as in the following example:

```
% nc run -f commandFile
```

All jobs submitted with this method share the same environment, the same resources, and are scheduled at the same priority level. Each job has its own ID.

Use Environments: `-e <env>`


1. Get a list of all available environments:

```
% vel  
vel: message: Environment directories:  
1 /<install_path>/local/environments  
1 . tcl BASE          UNIX utilities, X windows, and Flowtracer.  
1 . tcl D             Define vars: Usage: ves "+D(VAR1=value1,...)"  
1 . tcl DEFAULT      Just a name for whatever you already have.  
1 . csh SPICE        The Analog simulator SPICE3.  
1 . csh JAVA         JAVA Development Environment (1.2.2)
```

2. Use proper environment(s) to submit jobs, for example:

```
% nc run -e BASE sleep 10  
% nc run -e BASE+SPICE spice chip.spi
```

Use of Resources: -r <res1 res2 ...>

 **Note:** Since this option accepts multiple arguments, you need to terminate the resource list explicitly with another option, or "--" if no other option is needed.

1. Submit a job that only runs on Linux machine:

```
% nc run -r os=linux64 -- sleep 10
```

2. Submit a job that requires resources "License:spicy" and "hspice":

```
% nc run -r License:spicy -- sleep 10
```

Wait for Jobs

Waiting for jobs is especially useful for scripts. By default, `nc run` returns immediately after you submit a job. This allows you to submit multiple jobs at once. There could be times when you want to run jobs in sequence, in which case option `-w` is very useful. With this option, `nc run` waits for the job(s) to finish (Done or Failed).

```
% nc run -w sleep 10
Resources= linux
Env       = D(VOV_ENV_SOURCE=vnc_logs/envdixin36362.env)
Command  = vw vwrap sleep 10
Logfile  = vnc_logs/20021231/111314.21781
JobId    = 04211346
vnc: message: Scheduled jobs: 1          Total estimated time: 0s
```

Wait for Jobs and Show Log File of the last job: -wl

With this option, `nc run` waits for the job(s) and shows the log file of the last job.

```
% nc run -wl date
Resources= linux
Env       = D(VOV_ENV_SOURCE=vnc_logs/envdixin36362.env)
Command  = vw vwrap date
Logfile  = vnc_logs/20021231/111530.21819
JobId    = 04211392
nc: message: Scheduled jobs: 1          Total estimated time: 0s
<<<STARTING ON rhino>>>
Tue Dec 31 11:15:31 PST 2002
<<<END OF LOG>>>
<<<EXIT STATUS 0>>>
```

Please only use `-wl` for jobs where you are actively monitoring the output. Such jobs listen to the vovservers event stream and are called 'notify clients'. They are more resource-intensive than plain batch jobs.

Wait for Jobs Using `nc wait`

You can also wait for jobs using the command `nc wait`. We will cover that in next tutorial "Job Control".

Specify Name of logfile: **-l <logfile>**

The default logfile name has the form of `./vnc_logs/date/time`. You can explicitly specify the logfile of the jobs you submitted by this option, for example:

```
% nc run -l /home/john/logs/log1.log sleep 10
```

Warning: Conflicts may occur if same log file is used for multiple jobs. New users are not recommended to use this option.

Job Control

Accelerator provides several commands of job controls, including wait, stop and forget.

Wait for Jobs

Besides option `-w` and `-wl` with `nc run` command, you can also wait for job(s) to finish (Done or Failed) after they are submitted using the command `nc wait`. Here are some examples:

Get usage help

```
% nc wait -h
```

Wait for a job

```
% nc run sleep 10
Resources= linux
Env       = D(VOV_ENV_SOURCE=vnc_logs/envdixin36362.env)
Command   = vw vwrap sleep 10
Logfile   = vnc_logs/20021231/140024.23307
JobId     = 04213283
nc: message: Scheduled jobs: 1          Total estimated time: 0s

% nc wait 04213283
```

Wait for all jobs in current directory

```
% nc wait -dir .
nc: message: Job 04193913 is already FAILED
nc: message: Job 04193915 is already FAILED
nc: message: Job 04193917 is already FAILED
nc: message: Job 04193919 is not scheduled
nc: message: Job 04194308 is not scheduled
nc: message: Job 04211259 is already FAILED
nc: message: Job 04211268 is not scheduled
nc: message: Job 04213283 is already VALID
nc: message: Job 04213297 is already VALID
nc: message: Exiting with status 2 (Failed jobs)
```

Wait for all jobs using tool spice

```
% nc wait -select "tool==spice"
```

In the above example we use "Selection Rules" to perform a "wait" on those jobs that satisfy that rule. This could be useful for other commands as well, including `nc list`, `nc forget`, etc. Refer to the *Altair Accelerator User Guide* for more information.

Forget Jobs

The Accelerator server remembers the jobs you submitted for some configurable time. You can explicitly forget them by `nc forget` command, which will delete all job information from the server database.

Get a list of jobs

```
% nc list
04146420 Done      sleep 1      #in the form of "jobId status command"
```

```
04146425 Done      sleep 5
04146427 Running  sleep 10
04146429 Running  sleep 15
04146431 Queued   sleep 20
04146433 Queued   sleep 60
```

Forget some of them

```
% nc forget 04146420 04146425
nc: message: Forgetting 2 jobs
```

List jobs again (notice those two are gone)

```
% nc list
04146427 Done      sleep 10
04146429 Done      sleep 15
04146431 Done      sleep 20
04146433 Done      sleep 60
```

Forget all my jobs

```
% nc forget -mine
nc: message: Forgetting 4 jobs
```

Stop Jobs

A job can be stopped when it is either Running or Queued. Stopping a job does not forget it from the server database. "Running" jobs will exit, and "Queued" jobs will be dequeued when you stop them.

Stop some jobs

```
% nc run sleep 60
Resources= linux
Env       = D(VOV_ENV_SOURCE=vnc_logs/envdixin36362.env)
Command  = vw vwrap sleep 60
Logfile   = vnc_logs/20021231/140803.23386
JobId     = 04213393
nc: message: Scheduled jobs: 1          Total estimated time: 0s

nc stop 04213393
nc: message: Stopping RETRACING job 04213393
```

Stop all my jobs

```
% nc stop -mine
nc: message: Stopping RETRACING job 04146627
nc: message: Stopping RETRACING job 04146629
nc: message: De-queuing  job 04146631
nc: message: De-queuing  job 04146633
```

Rerun Jobs

The `nc rerun` command initiates the scheduling and execution of jobs that are already in the server database. By default, only the jobs that are Idle or Queued are affected by this command. If you want to force the rerunning of jobs that are either Done or Failed, use the option `-F`.

Rerun a "Done" job won't do anything

```
% nc rerun 04146622
nc: message: Not rerun: 04146622
```

Force rerunning a "Done" job

```
% nc rerun -F 04146622
nc: message: Job 04146622 is already VALID.
nc: message: Scheduled jobs: 1          Total estimated time: 1s
```

Rerun "Idle" jobs

```
% nc rerun 04146631 04146633
nc: message: Scheduled jobs: 1          Total estimated time: 0s
nc: message: Scheduled jobs: 1          Total estimated time: 0s
```

Get Detailed Information about a Job

The command `ncwx` displays information about a job.

Get Detailed Information About a Job

The command `nc getfieldwx getfield` also gives information about a job, but in an undecorated form that is in scripts.

```
nc: Usage Message

NC GETFIELD:
  Get one or all fields of one or more Accelerator jobs.  Specify the jobID
  or use '!' for the most recent job in the current working directory.

  If the -J jobName option is given, only the first match
  is reported.  If there is no match, an error is reported.

OPTIONS:
  -f field      -- Specify field when giving multiple jobIDs.
  -h           -- Help usage message. You can also get the usage message by
                specifying no option at all.
  -J JOBNAME   -- Find first job with given JOBNAME. The search is restricted
                to the jobs that belong to the current user. This is
                significantly more expensive than using jobIDs. Use
                sparingly.
  -s          -- Same as -showid.
  -sep STRING -- Use STRING as separator (default is a single space).
  -showid     -- Show jobId.
  -tab        -- Use a TAB character as separator.
  -v          -- Increase verbosity.

EXAMPLES:
% nc getfield -h
% nc getfield 01234455
% nc getfield 00123445 jobclass
% nc getfield ! status
% nc getfield -J JOBNAME
% nc getfield 01234455 0123458 -f jobclass
% nc getfield -s 01234455 0123458 -f jobclass
```

```
wx: Usage Message

WX GETFIELD:
  Get one or all fields of one or more WX jobs.  Specify the jobID
  or use '!' for the most recent job in the current working directory.

  If the -J jobName option is given, only the first match
  is reported.  If there is no match, an error is reported.

OPTIONS:
  -h          Show this help. You can also get the
                usage message by specifying no option at all.
  -v          Increase verbosity.
  -J JOBNAME Find first job with given JOBNAME
                The search is restricted to the jobs that
                belong to the current user.
```

```
This is significantly more expensive than
using jobIds. Use sparingly.
-f field      Specify field when giving multiple jobIDs
-showid      Show jobId
-s           Same as -showid
-sep STRING  Use STRING as separator (default is a single space)
-tab        Use a TAB character as separator.
```

EXAMPLES:

```
% wx getfield -h
% wx getfield 01234455
% wx getfield 00123445 jobclass
% wx getfield ! status
% wx getfield -J JOBNAME
% wx getfield 01234455 0123458 -f jobclass
% wx getfield -s 01234455 0123458 -f jobclass
```

Examples:

```
% nc getfield 00012345 jobclass
normal
% nc getfield 00012345 cputime
7.125
% nc getfield 00012345
... get list of all known fields (more than 100 of them)...
```

```
% wx getfield 00012345 jobclass
normal
% wx getfield 00012345 cputime
7.125
% wx getfield 00012345
... get list of all known fields (more than 100 of them)...
```

Monitor Jobs, Taskers and Resources

The activity of Accelerator can be monitored with a dialog.

The dialog is invoked with:

```
% ncwx monitor
```

The following is a list of the tabs available in the dialog:

TaskersGroups	The activity of tasker groups.
Taskers	The activity of taskers.
Taskers HW	The hardware offered by taskers.
Taskers Resources	The resources offered by taskers.
Who	Who is running jobs.
Running Jobs	The progress of running jobs.
Running Commands	The details of running commands.
Running Details	The details of running jobs.
Resources	The usage and availability of resources.
Queued Jobs	The jobs in the job queue.
Queue Buckets	The jobs in the job queue organized by groups of similar jobs (called 'buckets').
FairShare	The FairShare statistics.

TaskerGroups	Taskers	Tasker HW	Tasker Resources	Who	Running Jobs	Running Commands	Running Details	Resources	Queued Jobs	Queue Buckets	FairShare
Type:Name	Total	InUse	Rsrvd	ooQ	Available	%Util.	MapsTo				
1 License:AL002_HMPoll	unlim	0	0	0	unlim						
2 Limit:user0265_justo	1	0	0	0	1	99.12%					
3 License:AL002_HFSoli	100000	0	0	0	100000	0.00%					
4 License:AL003_HMPSD_	unlim	0	0	0	unlim						
5 Limit:user0418_justo	1	0	0	0	1	99.79%					
6 Limit:user0354_justo	1	0	0	0	1	97.64%					
7 Limit:user0100_justo	1	0	0	0	1	98.94%					
8 License:AL003_HMPSD_	unlim	0	0	0	unlim						
9 License:SolverNode	100000	0	0	0	100000	0.00%					
10 Limit:user1312_justo	1	0	0	0	1	98.16%					
11 License:AL003_HMPano	unlim	0	0	0	unlim						
12 License:AL003_SIMLAB	100000	0	0	0	100000	0.00%					
13 License:AL001_HMActi	unlim	0	0	0	unlim						

Figure 11:

Invoke the GUI

Job execution can be monitored with `nc guiwx gui`.

This command opens a monitoring tool; no interactive capabilities (such as configuration or running jobs) are provided. Interactive capabilities are available with `nc cmd vovconsolewx cmd vovconsole`.

nc gui

Show a grid view of the jobs in a specified set.

```
nc: Usage Message

NC GUI:
    Show a grid view of the jobs in a specified set.
USAGE:
    % nc gui [OPTIONS] &
OPTIONS:
    With no options, the GUI shows all jobs of the current
    user.

    -all
    -a                -- Show all jobs.
    -u <user>        -- Show jobs for specified user.
    -s <SETNAME>
    -set <SETNAME>
    -setname <SETNAME> -- Show specified set.
    -timeout <TIMESPEC> -- Stop async update after this time (default 2h).
    -submit
    -limitGui <N>    -- Override the limit of 3 max GUI per user.

    -batch <file>    -- Execute specified file after the GUI is ready

    -metrics
    -metricsConfig <file> -- Use specified metrics configuration file.
    -taskers
    -fontsize <size>   -- Specify the normal font size. Default is 10.
                       Legal range is 3 to 36.

    -title <title>   -- Choose title of X11 window.
    -iopprofile <jobId> -- Show job I/O profiling timeseries statistics
                       plots. The job must have been submitted with
                       the -iopprofile option. (preview feature)

EXAMPLES:
    % nc gui &                -- Show all my jobs
    % nc gui -all &           -- Show all jobs.
    % nc gui -set SomeSetName -- Show specified set.

    % nc gui -submit          -- Job submission dialog.
    % nc gui -limitGui 5      -- Allow you to run up to 5 "nc gui" (default 3)

    % nc gui -metrics &      -- Show the scheduler metrics.
```

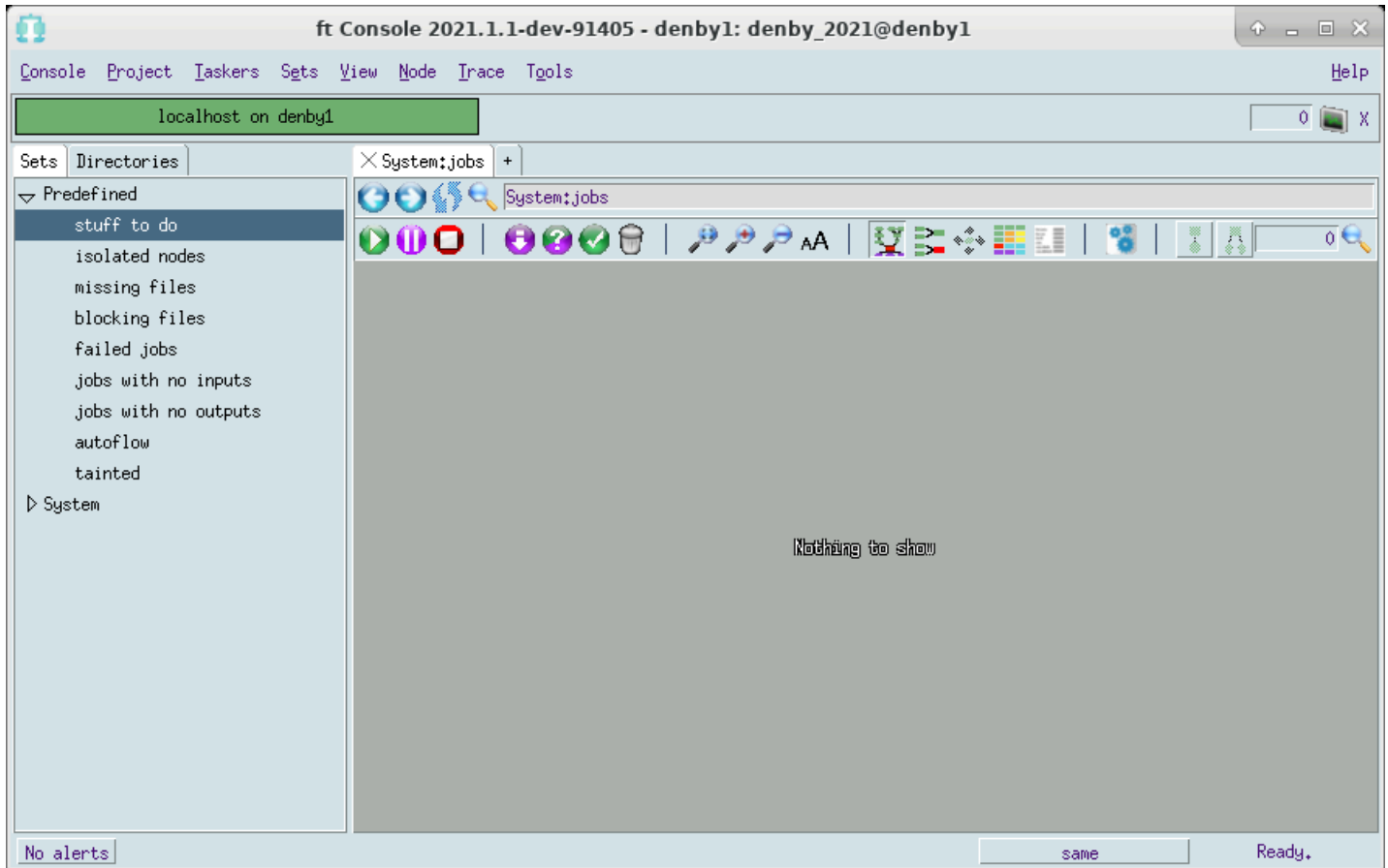


Figure 12: GUI that opens after entering nc cmd vovconsole &

Use the Web Browser

Find the URL for Accelerator

```
% nc cmd vovbrowser  
http://comet:6271/project
```

Use the Browser Interface

Enter the URL found above into your browser's location box. You will need to login unless your administrator has turned off authentication.

`http://your_host:your_port_number/project` is the home page of Accelerator. From this page, you can easily navigate to many other useful pages. Here we list some monitor pages that are similar to the ones from the GUI monitors.

Tasker page	From the home page, click Taskers
Workload pages (job queue and running jobs):	From the home page, click Workload For running jobs, under Workload, click Running jobs
Resources page	From the home page, click Resources
FairShare page	From the home page, under Workload, click Fairshare

Among the many useful pages, two report pages may be especially helpful.

Job queue report	From the home page, click Workload > Job Queue Reports
Resources reports	From the home page, click Resources > Resource Reports

Troubleshooting

This section covers typical problems that come up in Accelerator. The command `nc summary` will be useful here, as it tells you how many jobs are failed, queued, or idle, and what the queued jobs are waiting for. For example:

```
% nc summary
NC Summary For User bkring
TOTAL JOBS          0      Duration: 0s
Done                0
Idle                0
Queued              0
Running             0
Failed              0
```

My job won't start!

Often, your job won't start because it is waiting for a resource, usually a license, CPU, or memory.

To diagnose this, use the command:

```
nc info jobid
```

A job will only start if all resources requested by the job are available. If any resource is missing, the job will not start. You can look at the resources of all available vovtaskers to see if there is any that can run the job with the command:

```
nc host
```

Additionally, a vovtaskers must either be `READY` or `WRKNG` (have a free job slot) to accept jobs. Any other condition will prevent the vovtaskers from taking the job.

My job failed!

You can find the reason for job failure with the command:

```
nc info jobid
```

Some common failure conditions include:

- The job failure has nothing to do with Accelerator. Run the job without Accelerator to verify this.
- The job command doesn't exist, possibly because of a typo.
- You are using a wrong, nonexistent, or incomplete environment with the `-e`. In this case, `nc info jobid` will tell you that it cannot switch to the environment.
- You have failed to specify (or specified the wrong) architecture or memory usage. This can be done with the `-r` option. For example, `-r linux64` for Linux 64 bit or `-r RAM/2000` for 2GB of ram.

Altair Accelerator Administrator Tutorials

In the following tutorial, you will experiment with most issues that an Accelerator administrator will need to address, including starting/stopping the server, configuring FairShare, resources, taskers and environments.

This chapter covers the following:

- [Start a Test Queue](#) (p. 46)
- [Start/Stop Accelerator](#) (p. 49)
- [Browser-based Setup](#) (p. 51)
- [Configure Policy - FairShare and Other Parameters](#) (p. 52)
- [Advanced Policy Configuration](#) (p. 54)
- [Configure Resources](#) (p. 56)
- [Configure Security](#) (p. 58)
- [Configure Taskers](#) (p. 60)
- [Configure an Environment](#) (p. 65)
- [Logical Names \(Equivalences\)](#) (p. 70)
- [Resource Management](#) (p. 73)
- [Upgrade Accelerator](#) (p. 76)

You will start a test queue to make the experiment non-disruptive to the default Accelerator queue.

Also in This Tutorial

Start a Test Queue

At a given site, a single queue is recommended, such as a single Accelerator setup, by default called "vnc". Such a setup is called a *cluster* by other systems. The scheduler in Accelerator does not have queues in the sense used by other batch systems.

In this tutorial, you will start a temporary Accelerator queue for our testing. This allows you to experiment with most of the administration tasks without disturbing your production Accelerator queue.

Find the Server Working Directory

The Accelerator configuration directory for the default queue named "vnc" is in the product hierarchy. The server working directory for this queue is:

```
$VOVDIR/../../../../vnc/vnc.swd
```

This directory contains the server configuration files and server/tasker logs, etc.

Start a Queue

The command to start a queue is `ncmgr start`. By default, this command will start the default queue in the directory of `$VOVDIR/../../../../vnc`.

1. Get usage of this command:

```
vncmgr: Usage Message

USAGE:
    % ncmgr start [options]
OPTIONS:
    -h                This help.
    -force            Do not ask confirmation.
    -block            Do not return to the shell or command prompt
                    after starting. This is only useful, and
                    required, when starting Accelerator as a
                    Windows service.
    -port <port|mode> Specify port number, port number list (colon
                    separated) or port mode. Modes are:
                    automatic - hash queue name into port number,
                    do not start if port is
                    unavailable. The default queue
                    name 'vnc' hashes to port
                    6271.
                    any - hash queue name into port number, try
                    additional ports in increments of 1
                    until an available one is found. The
                    default queue name 'vnc' hashes to
                    beginning port 11437.
                    Default: any
    -webport <port|mode> Specify a dedicated web interface port for
                    HTTP and HTTPS protocols. This port must be
```

```
configured to enable REST API v3 interface,
to enable the dashboard web UI page,
and to enable SSL. A value of 0 directs
VovServer not to open a web interface port.
Specify port number, port number list (colon
separated) or port mode. Modes are:
automatic - hash queue name into port number,
do not start if port is
unavailable. The default queue
name 'vnc' hashes to web port
6271.
any - hash queue name into port number, try
additional ports in increments of 1
until an available one is found. The
default queue name 'vnc' hashes to
beginning web port 9695.
Default: Any
-webprovider <provider> Specify the provider for
HTTP and HTTPS protocols.
This must be either "internal" or "nginx".
Default: "internal"
-roport <port|mode> Specify read-only guest access web interface
port. A value of 0 disables this interface,
requiring all web interface users to log in.
Default: 0
-q, -queue <name> Name for queue (default is $NC_QUEUE if set,
and otherwise vnc).

-dir <dir> Directory of the server
(default $VOVDIR/../../vnc).
-dbhost <host> Host for database.
-dbroot <path> Path on database host for database files.
-dbport <port> Port of the database to listen for
connections.
-v Increase verbosity.
EXAMPLES:
% ncmgr start -port 6271
% ncmgr start -port 6271:6272:6273:any -force
% ncmgr start -q bigqueue -dir /remote/queues
```

2. Start the test queue in the home directory.
3. Name your queue vncdixin (make sure you pick a name that does not conflict with any existing queue). The first three letters of the queue name should be formed by prefixing your login with vnc
For example, if your login is 'danny', use the name 'vncdanny' for your queue.

```
+ start a shell on the machine where your NC vovserver should run
% cd # Go to your home directory
% mkdir ncadmin # Create a directory for our testing queue
% ncmgr start -dir ncadmin -queue vncdixin
message: Checking the license...
message: ... the license is good.
message: Starting NC
message: with name vncdixin
message: on host alpaca
message: in directory /home/dexin/ncadmin
message: as user dexin
Do you want to proceed? (yes/[no]) > yes
message: Updating config file '/remote/release/VOV/2013.09/linux64/local/
vncConfig/vncdixin.tcl' message: Waiting for server to be ready ...
message: Sanity check...
```

```
message: NC vncdexin@alpaca is ready.
```



Note: Make sure the directory `$VOVDIR/local/vncConfig` is writable to you, because a config file needs to be written in that directory to start a new queue. The Altair Accelerator installer should have set this.

Use a Specific Queue

Now that you have started a new queue, you have at least two Accelerator queues in your system. There are two ways to use a particular queue.

1. Use `-q` or `-queue` option:

```
# Submit a job to queue "vncdexin"
% nc -q vncdexin run sleep 60

# List my jobs in queue "vncdexin"
% nc -q vncdexin list

# Get info about queue "vncdexin"
% ncmgr info -queue vncdexin

# Reset all taskers for queue "vncdexin"
% ncmgr reset -queue vncdexin -taskers
```

2. Alternatively, you can set the environment variable `NC_QUEUE`:

You can also set an environment variable `NC_QUEUE` to the name of queue you want. Then you can execute all your `nc` and `ncmgr` commands in the context of that queue without having to use `-q` or `-queue` options.

```
% setenv NC_QUEUE vncdexin

# Now submit a job to queue "vncdexin"
% nc run sleep 61


# etc.
```

This way, you can easily switch between all the queues you have.

Start/Stop Accelerator

The commands involved in this tutorial are `ncmgr info`, `ncmgr start` and `ncmgr stop`.

You should use `ncmgr`, especially when starting Accelerator, because it checks for and applies any daemon and DB schema changes when moving to a new version.

 **Note:** Please remember to use the shell where you set the `NC_QUEUE` environment variable to the name of your training tutorial queue.

View Accelerator Status

Use command `ncmgr info` to get a summary information of all taskers and current running jobs.

```
% ncmgr info
ncmgr: message: NC vncdexin@alpaca
1 001 alpaca 1/0 146198 Unlim. IDLE
2 002 cheetah 1/0 235294 Unlim. IDLE
3 004 bison 1/1 99206 Unlim FULL
   2s: vw vwrap sleep 60 > vnc long
4 000 pluto 1/0 75312 Unlim. IDLE
5 003 rhino 1/0 181818 Unlim. IDLE
```

Start Accelerator

1. Execute `ncmgr` to see if Accelerator is already running.

```
% ncmgr start
ncmgr: USER ERROR: NC vncdexin@alpaca already running.
```

2. If not already running, start Accelerator. In production use, it is important to check the number of file descriptors available to `vovserver`.

```
% ncmgr start
ncmgr: message: Checking the license...
ncmgr: message: ... the license is good.
ncmgr: message: Starting NC
ncmgr: message: with name          vncdexin
ncmgr: message: on host            alpaca
ncmgr: message: in directory       /home/dexin/ftadmin
ncmgr: message: as user           dextrin
ncmgr: message: with             1024 file descriptors
Do you want to proceed? (yes/[no]) > yes
ncmgr: message: Waiting for server to be ready ...
ncmgr: message: Sanity check...
ncmgr: message: NC vncdexin@alpaca is ready.
```

Stop Accelerator

Run the following:

```
% ncmgr stop
ncmgr: message: Checking if NC vncdixin@alpaca is running...
ncmgr: message:
You are about to stop NC vncdixin@alpaca in directory /home/dexin/ftadmin
    Would you like to proceed (yes/[no]) ? > yes
ncmgr: message: Stopping taskers and server ...
ncmgr: message: NC vncdixin@alpaca has been stopped.
```

Restart Your Accelerator Queue

You should restart your queue so that it will be running for later exercises.

Run the following:

```
% ncmgr start
ncmgr: message: Checking the license...
ncmgr: message: Checking the license...
ncmgr: message: ... the license is good.
ncmgr: message: Starting NC
ncmgr: message:   with name      vncdixin
ncmgr: message:   on host        alpaca
ncmgr: message:   in directory   /home/dexin/ftadmin
ncmgr: message:   as user        dexin
    Do you want to proceed? (yes/[no]) > yes
ncmgr: message: Waiting for server to be ready ...
ncmgr: message: Sanity check...
ncmgr: message: NC vncdixin@alpaca is ready.
```

Browser-based Setup

Accelerator provides a simplified user-friendly browser-based setup, which is especially useful for new users. To use this browser-based setup page, please first make sure Accelerator is running.

Find the URL

First find the Accelerator URL, using `vovbrowser` or `vsi`:

```
% nc cmd vovbrowser
http://yourhost:6295/project
% nc cmd vsi
(more detailed output, including the Accelerator vovserver URL)
```

The setup script is available at the URL `/cgi/setup.cgi` (for example, in this case, it is `http://alpaca:6295/cgi/setup.cgi`).

Setup Using the Web Page

1. Follow the instructions on the web page to finish the basic setup.
2. On the Taskers setup page, try to add at least one tasker for each type (Server, Workstation, Offhours).
3. View the tasker statuses at URL `/taskers`.
4. Try out some test jobs.



Note: You need a working remote-shell setup to start non-local vovtaskers for this step.

Configure Policy - FairShare and Other Parameters

Accelerator has a time-windowing multi-level FairShare mechanism. This allocates CPU cycles to the fsgroups at each level according to the assigned weights of each group until the leaf nodes of the FairShare tree are reached. Each node of the FairShare tree may have a separate time window. The word 'fsgroup' is used as an abbreviation here.

To configure FairShare, use the `vovfsgroup` command.

Access to FairShare groups is controlled by ACLs (Access Control Lists), which means you can configure them so only designated users can submit jobs in an fsgroup.

Locate Server Configuration Directory to Find `policy.tcl`

The test queue was start in `vncdextrin` in directory `~/ncadmin`. So the server configuration directory for this Accelerator queue is `~/ncadmin/vncdextrin.swd`. By default, the server configuration directory is `$VOVDIR/./././vnc/vnc.swd`.

If you forget, you can find it out by this command:

```
% nc cmd vovserverdir  
/home/dextrin/ncadmin
```

In this case, you will find `policy.tcl` file at `/home/dextrin/ncadmin/vncdextrin.swd/policy.tcl`.

You will find other configuration files in the same directory, for example, `taskers.tcl`, `resources.tcl`.

Configure `policy.tcl` for FairShare

In the following example, two groups, `production` and `regression`, are configured.

Execute the following:

```
# in policy.tcl  
% vovfsgroup create /production -weight 400 -window 8h  
% vovfsgroup create /regression -weight 100 -window 8h
```

Two groups are defined: `/production` and `/regression`. When there are jobs from both the `production` group and `regression` group, the target share ratio of CPUs will be 400:100.

Save the New Configuration

Changes made by the `vovfsgroup` command are only in the `vovserver`'s memory until the next save, and may be lost if the `vovserver` is restarted before then.

Save the configuration to a file so it can be reloaded.

```
% nc cmd vovfsgroup genconfig myfsconfig.tcl
```

Test the New Fairshare Configuration

1. You use `-g` option in command `nc run` to submit a job from a particular group.

```
% nc run -g regression -f listOfJobs  
% nc run -g production -f listOfJobs
```

2. Use the Monitors GUI or browser page to monitor the dynamic changes of FairShare.
 - Use the monitors: use command `nc monitor` to bring up the monitors and click on the **FairShare** tab.
 - Use the browser: use command `nc cmd vovbrowser` to find the Accelerator URL, then find the FairShare page at URL `/cgi/fairshare.cgi`.

Server Configuration Parameters

Certain parameters of the Accelerator `vovserver` are also configurable by means of entries in the `policy.tcl` file. The [Server Configuration](#) page describes these in detail.)

Here are some examples:

```
# This is part of the policy.tcl file.  
set config(maxQueueLength)      8000  
set config(httpSecure)          1  
set config(saveToDiskPeriod)    2h;  
set config(autoLogout)          1h;    # Logout from browser interface  
  
# Used by Accelerator for autoforget.  
set config(autoForgetValid)      1h  
set config(autoForgetFailed)     2d  
set config(autoForgetOthers)     2d  
set config(autoRescheduleThreshold) 2s
```

Advanced Policy Configuration

Besides the vovserver configuration file `policy.tcl`, the behavior of job submission to Accelerator can also be controlled by the file `VOVDIR/local/vnc_policy.tcl`, which is used to define the following procedures:

<code>VncPolicyDefaultResources</code>	The default resources required by a job.
<code>VncPolicyValidateResources</code>	Make sure that the resource list for a job obeys any number of rules.
<code>VncPolicyDefaultPriority { user }</code>	Assign the default priority to a job based on the user.
<code>VncPolicyMaxPriority { user priority }</code>	Limit the priority based on the maximum allowed to the user.

In this tutorial, you will configure `VncPolicyDefaultResources` and `VncPolicyValidateResources`.

Configure Default Resources

By default, Accelerator takes machine architecture as the resource of the job that you submit from a particular machine. This is controlled by the default setting of `VncPolicyDefaultResources`:

```
proc VncPolicyDefaultResources {} {  
    global env  
    return "$env(VOVARCH)"  
}
```

To change this behavior, you can create or edit the file `VOVDIR/local/vnc_policy.tcl`, and add or edit the procedure that follows. This procedure will set the default resources to be the architecture and 50mb of memory. If you have more than one Accelerator setup, you can place the `vnc_policy.tcl` file in the `.swd` and it will apply only to that one.

```
proc VncPolicyDefaultResources {} {  
    global env  
    return "$env(VOVARCH) RAM/50"  
}
```

Enforce Job Resource Rules

You can also enforce some rules of job resources by overriding the procedure `VncPolicyValidateResources`. Here is the default behavior:

```
proc VncPolicyValidateResources { resList } {  
    return $resList  
}
```

But this process does nothing. Try the process described below:

To enforce a rule so that all jobs require a minimum RAM of 512 MB, create or edit the file `$VOVDIR/local/vnc_policy.tcl` and add or edit `VncPolicyValidateResources` procedure as follows:

```
proc VncPolicyValidateResources { resList } {  
  #  
  # This policy adds a minimum RAM requirement  
  # for all submitted jobs.  
  #  
  if [regexp "RAM/" $resList] {  
    # Already a RAM constraints.  
  } else {  
    lappend resList "RAM/512"  
  }  
  return $resList  
}
```

Configure Resources

Accelerator includes a sophisticated subsystem for the management of computing resources, which allows the design team to take into account all sorts of constraints regarding hardware and software resources, as well as site policy constraints. This mechanism is based on:

- The resources required by jobs
- The resources offered by taskers
- The ResourceMap, as described in the file `resources.tcl`

Next, you will configure the resource map in `resources.tcl`.

Find and View the resources.tcl File

You can find this file in the server configuration directory. For default vnc queue, it is `$VOVDIR/./././vnc/vnc.swd`. For the test queue, it is in directory `~/ncadmin/vncdexin.swd/`.

```
% cd ~/ncadmin/vncdexin.swd
% vi resources.tcl ; # Use the editor of your choice
```

```
# ... here we only show part of this file ...
vtk_resourcemap_set PRIORITY_LOW      1
vtk_resourcemap_set PRIORITY_NORMAL  10
vtk_resourcemap_set PRIORITY_HIGH     20
vtk_resourcemap_set PRIORITY_TOP      UNLIMITED
```

With above default configuration, there will be at most 1 low priority job running at any time, 10 for normal, 20 for high, and any number of top priority jobs could be running.

1. Run a simple test to verify that the above information is correct:

```
% nc run -f $VOVDIR/training/vnc/cmdlist.unix
```

2. You can use the Monitors window to monitor the "Running Jobs" and "Resources" to see the running jobs and resources usage. You can also use browser to get similar information from "Running Jobs" page and "Resources" page.

Configuration Examples

Example 1

For example, if you decide that no more than 4 normal priority jobs should be running at any time, you can edit `resources.tcl` and modify the value for normal priority to 4.

```
# ... here we only show part of this file ...
vtk_resourcemap_set PRIORITY_LOW      1

# Now we change this value to 4
vtk_resourcemap_set PRIORITY_NORMAL  4
```



```
vtk_resourcemap_set PRIORITY_HIGH 20  
vtk_resourcemap_set PRIORITY_TOP UNLIMITED
```

Save your change and do a reread and test the new configuration:

```
% nc cmd vovproject reread  
% nc run -f $VOVDIR/training/vnc/cmdlist.unix
```

Example 2

You can configure your resources similarly. For example, you have 10 calibre license, you can configure this by adding the following line in `resources.tcl`:

```
# In resources.tcl  
vtk_resourcemap_set calibre_license 10
```

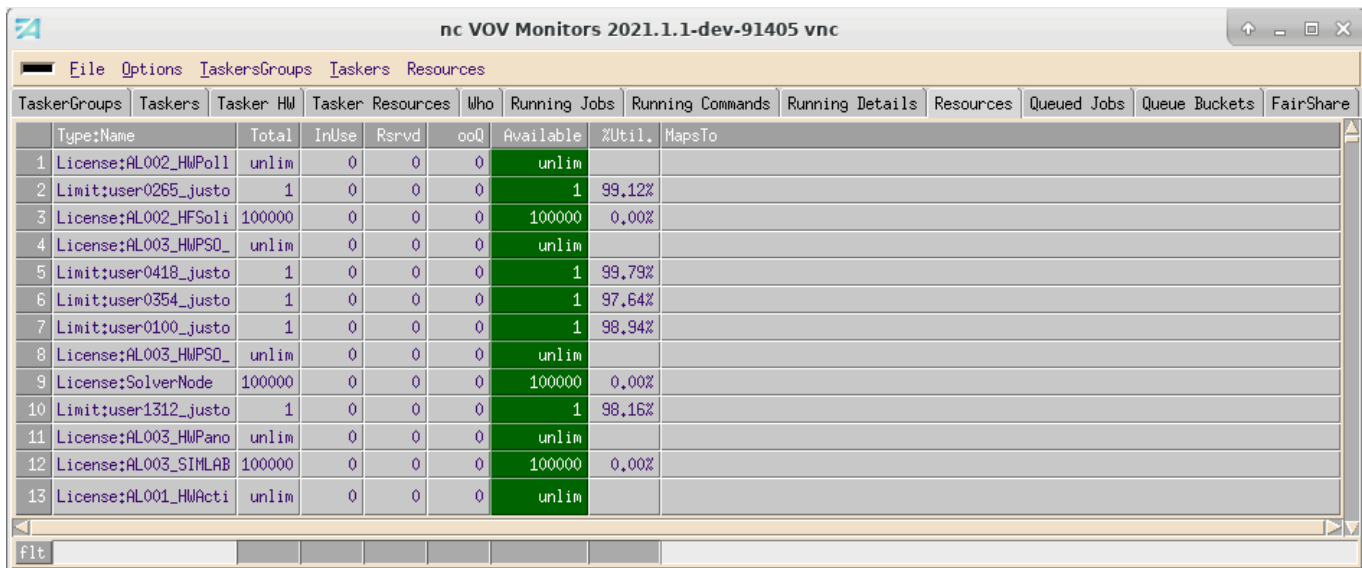
You can also associate a resource to other resource(s). For example, you have 4 licenses of spice that are only available on Linux. You can configure this by add the following line in `resources.tcl`:

```
# In resources.tcl  
vtk_resourcemap_set hspice_license 4 linux
```

Resource Monitor

To monitor resource activity, call a GUI monitor with the command:

```
nc mon
```



The screenshot shows a window titled "nc VOV Monitors 2021.1.1-dev-91405 vnc" with a menu bar (File, Options, TaskersGroups, Taskers, Resources) and a toolbar. Below the toolbar is a table with columns: TaskerGroups, Taskers, Tasker HW, Tasker Resources, Who, Running Jobs, Running Commands, Running Details, Resources, Queued Jobs, Queue Buckets, and FairShare. The table contains 13 rows of resource data.

TaskerGroups	Taskers	Tasker HW	Tasker Resources	Who	Running Jobs	Running Commands	Running Details	Resources	Queued Jobs	Queue Buckets	FairShare
1	License:AL002_HWPoll	unlim	0	0	0	unlim					
2	Limit:user0265_justo	1	0	0	0	1	99.12%				
3	License:AL002_HFSoli	100000	0	0	0	100000	0.00%				
4	License:AL003_HWPSO_	unlim	0	0	0	unlim					
5	Limit:user0418_justo	1	0	0	0	1	99.79%				
6	Limit:user0354_justo	1	0	0	0	1	97.64%				
7	Limit:user0100_justo	1	0	0	0	1	98.94%				
8	License:AL003_HWPSO_	unlim	0	0	0	unlim					
9	License:SolverNode	100000	0	0	0	100000	0.00%				
10	Limit:user1312_justo	1	0	0	0	1	98.16%				
11	License:AL003_HWPano	unlim	0	0	0	unlim					
12	License:AL003_SIMLAB	100000	0	0	0	100000	0.00%				
13	License:AL001_HWActi	unlim	0	0	0	unlim					

Figure 13:

Configure Security

Accelerator has 4 privilege levels: READONLY, USER, LEADER, ADMIN.

For detailed information about security, please refer to [Security](#).

Locate the Security Configuration File: security.tcl

This file is in the server configuration directory, default \$VOVDIR/../../vnc/vnc.swd/security.tcl and in our test setup, that is ~/ncadmin/vncdexin.swd/security.tcl.

Security Configuration Examples

Least Restrictive Security

The least restrictive security grants everybody full access from any host. This should not be used in production.

```
# All users (+) are administrators from all hosts (+).  
vtk_security + ADMIN +
```

Alternatively, a VovUserGroup may be utilized, to assign individuals in a group the ADMIN privilege.

```
# Members of mygroup are administrators from all hosts (+).  
vtk_security -group mygroup ADMIN +
```

Most Restrictive Security

```
# No rule defined gives only the owner of the project ADMIN privileges  
# on the server host.
```

Typical Case

The following example shows a typical security file, in which different privileges are granted to different users. Also notice the use of variables and VovUserGroups in this example.

In the example, mary is an administrator for any host, and dan is an administrator only for reno and milano. The user pat is a LEADER for her machine elko, and fred has USER privileges for 4 machines listed in the variable \$allhosts. Members of the VovUserGroup "operators" have ADMIN rights on \$allHosts.

```
set servers          { reno milano }  
set allhostsset     { reno milano elko tahoe}  
  
vtk_security mary    ADMIN    +  
vtk_security john    ADMIN    tahoe  
vtk_security dan     ADMIN    $servers  
vtk_security pat     LEADER   elko  
vtk_security fred    USER    $allhosts
```

```
vtk_security -group operators ADMIN $allHosts
```

Configure Taskers

The file `taskers.tcl` describes the vovtaskers for Accelerator.

vovtasker Configuration

You can find the file `taskers.tcl` in server configuration directory, default at `$VOVDIR/../../vnc/vnc.swd/taskers.tcl`, and in our test case at `~/ncadmin/vncdexin.swd/taskers.tcl`.

This file is based on two commands (Tcl procedures), `vtk_tasker_define` and `vtk_tasker_set_default`:

```
# Set default behavior of vovtaskers
vtk_tasker_set_default [options]

# Define a vovtasker
vtk_tasker_define hostname [options]
```

If you set some options with `vtk_tasker_set_default` command, all the following `vtk_tasker_define` commands will use those options implicitly, and options set explicitly in `vtk_tasker_define` overwrite those set in `vtk_tasker_set_default`.

For example, to declare 3 vovtaskers on the hosts `apple`, `orange`, and `pear`, use:

```
# In taskers.tcl
vtk_tasker_define apple
vtk_tasker_define orange
vtk_tasker_define pear
```

or some equivalent code:

```
# In taskers.tcl
foreach host {apple orange pear} {
    vtk_tasker_define $host
}
```

To understand the use of `vtk_tasker_set_default`, define 3 vovtaskers as follows:

```
vtk_tasker_define apple -resources "@STD@ big_memory" -CPUS 2
vtk_tasker_define orange -resources "@STD@ big_memory" -CPUS 2
vtk_tasker_define pear -resources "@STD@ big_memory" -CPUS 4
```

`vtk_tasker_set_default` does the following:


```
vtk_tasker_set_defaults -resources "@STD@ big_memory" -CPUS 2
vtk_tasker_define apple
vtk_tasker_define orange
vtk_tasker_define pear -CPUS 4
```

Tasker Configuration Examples

The default `taskers.tcl` provides pretty good examples of configuring vovtaskers. Most of the time, you just need to plug in some host names in brackets to define vovtaskers. For example, here is one piece of code in `taskers.tcl`.

```
# ADD THE NAMES OF THE COMPUTE SERVERS TO THE FOLLOWING LIST
set mainComputeServers {}

foreach host $mainComputeServers {
    vtk_tasker_define $host -resources "VovResources::Standard @RAMTOTAL@ @SWAPFREE@"
}
```

 **Note:** `-resources "VovResources::Standard"` is equivalent to `-resources"@STD@"`

To add some "Server" class vovtaskers, you just need to add the names of those hosts into the list `mainComputerServers`, like the following:

```
set mainComputeServers { apple orange pear }
```

You can certainly add or modify the vovtasker definition as you want (subject to license restriction). For example, you have many dual CPU machines, and you would like to make the *maxload* of these vovtasker machines bigger, say, equal to 1.5 times of the CPUs, for example 3.0. Then you can do this:

```
set myDualCpuServers {}

foreach host $myDualCpuServers {
    vtk_tasker_define $host -maxload 3.0 -resources "VovResources::Standard
@RAMTOTAL@ @SWAPFREE@"
}
```

Add Workstation/Offhours vovtaskers

```
# Add a Workstation vovtasker that will only start to accept
# job after 10 minutes of ilde and will only accept jobs
# with expected duration no longer than 5 minutes
vtk_tasker_define ftcsun44 -resources "VovResources::Workstation -minIdle 10m -
maxtime 5m @RAMTOTAL@"

# Add a Offhours vovtasker that's only available from 7pm to 6am
# every weekday and on weekends
vtk_tasker_define ftcsun66 -resources "VovResources::Offhours"
```

Start Newly Defined vovtaskers

Like other configurations, Accelerator will pick up the changes in `taskers.tcl` when the server is stopped and restarted. This is often too disruptive and not favorable, especially when there are jobs running in Accelerator. Here are some other ways to apply your changes:

1. To start the vovtaskers that you just defined, use command:

```
% ncmgr reset -taskers
```

2. To start vovtaskers that you just modified, you must stop and start them.

Start/Stop vovtaskers (Advanced)

You can start/stop vovtaskers from the GUI and the browser.

The following is an advanced command of FlowTracer vovtaskermgr and how to run any command in the context of Accelerator using `nc cmd`.

```
# List all vovtaskers defined in taskers.tcl
# Also checks the taskers.tcl file for syntax errors
% nc cmd vovtaskermgr list

# Start all vovtaskers defined in taskers.tcl
% nc cmd vovtaskermgr start

# Start some vovtasker(s)
% nc cmd vovtaskermgr start tasker1 tasker2

# Stop all vovtaskers
% nc cmd vovtaskermgr stop

# Stop some vovtasker(s)
% nc cmd vovtaskermgr stop tasker1 tasker2

# Show detail information of all vovtaskers
% nc cmd vovtaskermgr show

# Get usage of all vovtaskermgr commands
% nc cmd vovtaskermgr
```

Start a vovtasker from Command Line (Advanced)

You can also start a vovtasker on the fly from the command line using the `vovtasker` binary. This can sometimes be handy, for example, for debugging. This is the command that Accelerator uses to start the vovtaskers after it reads the `taskers.tcl` configuration file.

1. To start a vovtasker on a particular host, you need to go to that host and use command `nc cmd vovtasker` with appropriate options, which are similar to the options of `vtk_tasker_define`.
2. Try the following examples and monitor the vovtaskers using the Monitor GUI or browser Tasker page.
 - Get usage of this command:

```
% nc cmd vovtasker
usage: vovtasker [-A startupLogFile] [-a name] [-b capabilities] [-B]
                [-c coefficient] [-C cpus] [-d] [-D integer] [-e reserveExpr]
                [-E] [-f tclfile] [-F <file>]
                [-g taskergroup] [-G group] [-h host] [-H HEALTHCHECKFLAGS]
                [-I 0|1] [-I tclfile] [-j] [-k d|n|v] [-l rootofDailyLogFile]
```

```
[-L <loadSensor>] [-m <integer>][-M max_load] [-n <integer>]
[-N] [-o local resource] [-p project]
[-P <double>] [-r resources] [-R resources] [-s] [-S
resources] [-t timeout] [-T capacity/max_capacity]
[-U updateInterval] [-v number] [-V ncName@ncHost[:port]]
[-w WX properties] [-z <timeSpec>] [-Z <timeSpec>] [-g name]
[-u name]

-A: The name of the startup log file
-a: Name this tasker. The name may contain only letters, numbers,
dash(-) and underscore(_), or the expressions @HOST@ and @PID@ that get
expanded on the fly
-b: Comma-separated list of capabilities, case insensitive:
    symbolic: FULL NC LM
    normal: PROCINFO NC LM
    short: P N X R
-B: Show BPS tasker objects. Default to not show.
-c: Tasker coefficient (positive, default 1.0)
-C: Number of CPU's in this machine (automatic on win64). Use 0
to specify default value

...OMITTED...
```

- Start a normal vovtasker (with all default settings):

```
% nc cmd vovtasker -N
vovtasker Jan 10 13:44:42
Copyright © 1995-2020, Altair Engineering Linux/7.1 Jan 10 2019 10:00:59
vnc@alpaca
vovtasker Jan 10 13:44:42 Test 1: INTEGER OPS W= 1.00 Reps= 500 T= 5.00ms
vovtasker Jan 10 13:44:42 Test 1: DOUBLE OPS W= 1.00 Reps= 25 T= 10.00ms
vovtasker Jan 10 13:44:42 Test 1: CHAR OPS W= 0.10 Reps= 10 T= 10.00ms
vovtasker Jan 10 13:44:42 ---- Weighted time: 16.00ms
vovtasker Jan 10 13:44:42 Test 2: INTEGER OPS W= 1.00 Reps= 500 T= 5.00ms
vovtasker Jan 10 13:44:42 Test 2: DOUBLE OPS W= 1.00 Reps= 25 T= 10.00ms
vovtasker Jan 10 13:44:42 Test 2: CHAR OPS W= 0.10 Reps= 10 T= 20.00ms
vovtasker Jan 10 13:44:42 ---- Weighted time: 17.00ms
vovtasker Jan 10 13:44:42 Test 3: INTEGER OPS W= 1.00 Reps= 500 T= 5.00ms
vovtasker Jan 10 13:44:42 Test 3: DOUBLE OPS W= 1.00 Reps= 25 T= 10.00ms
vovtasker Jan 10 13:44:42 Test 3: CHAR OPS W= 0.10 Reps= 10 T= 20.00ms
vovtasker Jan 10 13:44:42 ---- Weighted time: 17.00ms
vovtasker Jan 10 13:44:42 Best weighted time: 16.00ms
```

- Start a vovtasker with name "myvovtasker", max load 4.0, and offers standard resources "@STD@" and resource "special_license":

```
% nc cmd vovtasker -a myvovtasker -M 4.0 -r "@STD@ special_license"
vovtasker Jan 10 13:51:00
Copyright © 1995-2020, Altair Engineering Linux/7.1 Jan 10 2019 10:00:59
vnc@alpaca
vovtasker Jan 10 13:51:00 Test 1: INTEGER OPS W= 1.00 Reps= 500 T= 5.00ms
vovtasker Jan 10 13:51:00 Test 1: DOUBLE OPS W= 1.00 Reps= 25 T= 5.00ms
vovtasker Jan 10 13:51:00 Test 1: CHAR OPS W= 0.10 Reps= 10 T=
20.00ms
vovtasker Jan 10 13:51:00 ---- Weighted time: 12.00ms
vovtasker Jan 10 13:51:00 Test 2: INTEGER OPS W= 1.00 Reps= 500 T= 5.00ms
vovtasker Jan 10 13:51:00 Test 2: DOUBLE OPS W= 1.00 Reps= 25 T= 5.00ms
vovtasker Jan 10 13:51:00 Test 2: CHAR OPS W= 0.10 Reps= 10 T=
20.00ms
vovtasker Jan 10 13:51:00 ---- Weighted time: 12.00ms
vovtasker Jan 10 13:51:00 Test 3: INTEGER OPS W= 1.00 Reps= 500 T= 5.00ms
vovtasker Jan 10 13:51:00 Test 3: DOUBLE OPS W= 1.00 Reps= 25 T= 5.00ms
```

```
vovtasker Jan 10 13:51:00 Test 3: CHAR OPS    W= 0.10 Reps= 10    T=  
20.00ms  
vovtasker Jan 10 13:51:00 ---- Weighted time: 12.00ms
```


Configure an Environment

VOV includes a sophisticated subsystem to manage environments. In Accelerator, an environment is a named collection of environment variables and their values. Many tools expect certain environment variables to be set for correct operation, and the variable `PATH` is used by most UNIX shells to locate executable programs.

A VOV environment definition consists of three scripts defined in either Tcl, C-shell, or Bourne-shell syntax. Environments that will be used on both UNIX and Windows must be written in Tcl. The names of these scripts must follow a naming convention. For example, an environment named `ENV` would use these three scripts:

- `ENV.start.tcl`
- `ENV.end.tcl`
- `ENV.doc`

The 'start' script is used when entering the environment, the 'end' script is used when leaving the environment, and the 'doc' script contains a one-line summary of the environment's purpose.

VOV provides commands for using environments from the command line:

<code>vel</code>	List available environments
<code>vesENV</code>	Switch to environment ENV
<code>vep</code>	Change prompt to show environment name
<code>veprestore</code>	Return to original prompt

Find and View Example Environment Scripts

It is interesting and instructive to examine some of the environment definitions which are shipped with VOV. One of the most important is the `BASE` environment.

The most common place for environment scripts is `$VOVDIR/local/environments`, but you can also specify additional directories where VOV will search. We will discuss this in more detail later in the lab.

The `BASE` definition is in the 'local' directory of the VOV software installation, at `$VOVDIR/local/environments/BASE.start.tcl`

```
% pushd $VOVDIR/local/environments
% view BASE.start.tcl

# --
# -- Completely reset the environment to bare bones.
# -- We assume that VOVDIR and VOVARCH are set.
# --

set VOVDIR $env(VOVDIR)

if { $::tcl_platform(platform) eq "windows" } {
#
```

```

# --
#
nt_preprocess_env
# set vovenv(debug) 1 set WINDIR $env(WINDIR)
setenv PATH "$WINDIR\\system32;$WINDIR"

set MKSDIR "c:/mksdemo"
if [file exists $MKSDIR] {
    setenv ROOTDIR $MKSDIR
    vovenv PATH ";" APPEND $MKSDIR/mksnt
}
#
# Depending on the installation, the binaries can be in
# different directories.
#
foreach b { bin bin-0 bin-g bat local/bat } {
    set p "$VOVDIR/$b"
    #
    # Eliminate double // as in w://bat
    #
    regsub -nocase {^([a-z])://} $p {\1://} p

    if [file exists $p] {
        vovenv PATH ";" APPEND $p
    }
}

set version [exec voversion]
vovenv PATH ";" PREPEND c:/temp/vov/execache$version

# vovenv PATH ";" PREPEND c:/temp/vov/execache

} else {
    #
    # -- Unix BASE environment.
    #

    setenv PATH ""
    foreach b { bin bin-0 bin-g } {
        if [file exists $VOVDIR/$b] {
            vovenv PATH : APPEND $VOVDIR/$b
            break
        }
    }
    vovenv PATH : APPEND $VOVDIR/scripts
    vovenv PATH : APPEND $VOVDIR/local/scripts

    setenv MANPATH ""
    foreach mp [list /usr/man /usr/share/man /usr/local/man /usr/openwin/man $VOVDIR/man]
    {
        if [file isdirectory $mp] {
            vovenv MANPATH : APPEND $mp
        }
    }

    setenv LD_LIBRARY_PATH ""
    foreach lp [list /lib /usr/lib /usr/local/lib /usr/openwin/lib $VOVDIR/lib] {
        if [file isdirectory $lp] {
            vovenv LD_LIBRARY_PATH : APPEND $lp
        }
    }

    foreach xapp { /usr/openwin/lib/app-defaults /usr/lib/X11/app-defaults } {

```

```
if [file isdirectory $xapp ] {
    vovenv XFILESEARCHPATH : APPEND "$xapp/%N"
}
}

foreach p {
    /bin /usr/ucb /usr/bin /usr/local/bin
    /usr/X11/bin
    /usr/dt/bin
    /usr/openwin/bin /usr/bin/X11
} {
    if [file isdirectory $p] {
        vovenv PATH : APPEND $p
    }
}
}
```

The BASE environment includes just the regular system commands and VOV. This is an example of a complex environment setup script; most of the ones you will need to define will be much simpler.

A simple test to verify that:

```
% vep
```


Notice that the prompt has changed to show that you are in the DEFAULT environment. This is the environment which is defined by your regular system setup files, for example, `.cshrc`.

```
% ves BASE
% printenv PATH
```

Notice that the PATH environment variable is now (probably) much shorter, and contains the VOV software directories.

```
% veprestore
```

Notice that the prompt has been changed back to the original.

 **Note:** This only changes the prompt. Your shell still has whatever environment was set most recently using `vep`.

Add Additional Environment Directories

When you are testing and developing environments, or when you want to have project-specific ones which are not put in the system wide directory at `$VOVDIR/local/environments`, you can use the environment variable `VOV_ENV_DIR` to specify additional directories.

You will use this capability in the tutorial, so that you don't need to put our environment scripts in the system's shared environment area. You should have a `~/ncadmin` directory from the previous exercises, which contains the server working directory of your private Accelerator queue.

```
% setenv VOV_ENV_DIR ~/ncadmin/vnc-user-name.swd/environments
% pushd $VOV_ENV_DIR
```



CAUTION: The `enviroments` directory is generated by setup, but any user-made env-scripts placed in the directory will not be used unless `VOV_ENV_DIR` includes it.

Configuration Examples

In this example, you will create an environment which adds your personal bin directory to the path. You will put the definitions in your queue's environment directory.

Use Tcl for the environment scripts because it is more powerful than C-shell, and platform-independent.

Example 'start' script for JOHN environment

```
# add john bin directory to path
vovenv PATH : PREPEND /home/john/bin
```

This script adds the element `/home/john/bin` to the beginning of the `PATH`. There is also an `APPEND` operator which adds to the end of the `PATH`. If the exact element is already present in the path, it is left undisturbed. This avoids `PATH` overflow in C-shell.

Example 'end' script for JOHN environment

```
# remove john bin directory from path
vovenv PATH : DELETE /home/john/bin
```

This script removes the element `/home/john/bin` from the `PATH`.

Example 'doc' script for JOHN environment

```
prepend /home/john/bin to PATH
```

This file contains a one-line summary which is presented by the `vel` command.

Use the above examples to create similar files in your queue.

Compose Good Environment Scripts

There are several characteristics of a 'good' environment setup.

minimal

A minimal environment only sets the variables needed to accomplish a specific purpose.

composable

A composable environment respects pre-existing values of variables, and adds to them, rather than overwriting them. For example, when modifying `LM_LICENSE_FILE`, use `PREPEND` and `APPEND` rather than simply setting it.

This allows you to use commands like `ves BASE+CADENCE+NASSDA` and have both Cadence and Nassda tools work.

reversible

A good environment definition uses the 'end' script to revert what it set in the 'start' script, so that the environment does not gradually become polluted with obsolete values.

Environments are Cached on the vovtasker

To provide very low job dispatch latency, Accelerator caches the eight most-recently used environments in the vovtasker process. This means that jobs which run in any of those environments may start immediately without sourcing the environment definition script. It also means that if you change an env-script, you must tell the vovtaskers by using the `vovtaskermgr refresh` subcommand.

Summary

- Most environments are defined by scripts stored in `$VOVDIR/local/environments`
- Environment scripts are in Tcl or C-shell syntax
- You can use the `VOV_ENV_DIR` environment variable to specify additional directories to search for environment scripts.
- Environments are cached on the vovtasker, and refreshed using `ncmgr reset -taskers`, or `nc cmd vovtaskermgr refresh`


Logical Names (Equivalences)

Most sites set up their machines so that they all have uniform mountpoints and view of the file systems. A common scheme is to place all user home directories under a single parent directory, often called /home. The actual file system is mounted using automount, and the physical file system is located by an NIS map.

Unlike some other batch/network computing systems, Accelerator does not require that all the compute machines have a uniform set of mountpoints. Accelerator can use what are called logical names or equivalences, to locate the data.

The default `equiv.tcl` file which is shipped with the software defines a logical name for the path to VOVDIR.

The `equiv.tcl` file in some versions also defines a logical name HOMES for whatever the parent of your home directory is set in the environment variable HOME. The default one shipped with an early version did not, so we will show the steps to define a logical name using the browser-based setup.

 **Note:** When doing the exercises, remember to give these commands from a shell where you have set the environment variable NC_QUEUE to the name of your training tutorial queue. Otherwise, you will need to use the `-queue` option.

Define a Logical HOME Directory Name

1. Use the browser-based setup to view the logical names that are defined. First, find the URL of the queue's server, and connect a web browser to it.

```
% nc cmd vovbrowser --> http://host:port/project
```

```
% firefox http://host:port/cgi/setup.cgi
```

The initial page will have a link on the left called FileSystems.

2. Left-click on this link to display the logical names defined for the queue. You will see VOVDIR and VNCSWD.
3. Run a job in your home directory.

```
% cd  
% nc run sleep 10
```

This will often run, because the default is to require the architecture from which the job was submitted as a resource, so the job may run on your machine.

4. Now try the job, but request an architecture different from the machine on which you are working. This job will fail, because the directory on the remote machine does not have a logical name.

```
% nc run -r linux64 -e BASE pwd
```

```
Resources= linux64 Env = BASE  
Command = vw vwrap pwd
```

```
Logfile = vnc_logs/20030108/143125.27906 JobId = 00000021  
vnc: message: Scheduled jobs: 1 Total estimated time: 0s
```

```
% nc list  
00000011 Done sleep 10  
00000021 Failed pwd
```

```
% nc info -l 21  
  
Log file is: '${VNCSWD}/vncjohn.swd/vnc_logs/20030108/143125.27906'  
vnc: message: File ${VNCSWD}/vncjohn.swd/vnc_logs/20030108/143125.27906 does not  
exist (yet)  
    (mapped to /usr2/home/john/ncadmin/vncjohn.swd/  
vnc_logs/20030108/143125.27906)
```

5. Run the job in the /tmp directory.

/tmp exists on all UNIX machines, but is not usually exported to the network.

```
% cd /tmp  
% nc run sleep 10
```

This fails, because /tmp is not a network path.

```
% nc run sleep 10  
  
WARNING:  
  The path to the current directory is not 'LOGICAL',  
  i.e. it does not begin with a '$' sign.  
EXPLANATION:  
  The current directory  
  /tmp  
  may not be a valid directory path for all hosts in the network.  
OPTIONS:  
  (1) Continue.  
      By choosing this option, you assert that the path  
      is valid everywhere. If it is not, the job is likely to fail,  
      because the remote host cannot reach the current directory.  
      This option causes the creation of a flag file called .vnc which has  
the  
      purpose of avoiding the repetition of this question for this directory  
and  
      its subdirectories  
  (2) Abort.  
      Please ask your NetworkComputer administrator to change the equiv.tcl  
file to  
      define the rules that give a logical name to the current directory.
```

If you respond **2**, the job will not be submitted.

If you respond **1**, a file named .vnc will be created in the directory, and the job will be submitted. In the future, you will not receive the warning so long as the .vnc file exists.

The Altair Accelerator software internally uses logical names to refer to files.

Advanced Commands

1. Resource the cache file.

```
ls ~/ncadmin/vncjohn.swd/equiv.caches  
cayman pluto  
cayman abbcanova@cayman DEFAULT+SUPPORT html/ftnctraining > cat !$/pluto
```

```
cat ~/ncadmin/vncjohn.swd/equiv.caches/pluto  
VOVDIR /remote/release/VOV/7.0u2/linux64  
VOVDIR /remote/release/VOV/7.0u2/linux64/../../common
```

2. Run the vovequiv command.

```
% vovequiv -p . # show the logical name for this directory
```


Resource Management

This exercise will walk you through some of the more advanced and complex issues of resource management in Accelerator. You should already be familiar with Accelerator and basic resource management, covered in the [Configure Resources](#) tutorial.

With Accelerator, Altair Accelerator includes a simplified version of our Monitor product, called Monitor-basic, which interfaces between NC and licensing systems like FlexNet Publisher. Unlike the full LM product, LM-basic does not do history or denials.

Review

The following topics are review from the [Configure Resources](#) tutorial:

- Find and view `resources.tcl` file
- The `vtk_resourcemap_set` procedure
- Static Resource (PRIORITY_LOW) configuration example
- Configure resources map on the fly using `vtk_resourcemap_set`

Topics of this tutorial include:

- Monitor-basic setup
- `vtk_flexlm_monitor` procedure
- Browser-based setup
- Resource throttling

In this tutorial, we will start LM-basic, configure it, and use it to monitor licenses. There are two things necessary to monitor FlexNet Publisher features:

1. Configure and start the LM-basic `vovserver`.
2. Accept the default `vtk_flexlm_monitor_all`, or add specific `vtk_flexlm_monitor` statements to NC's `resources.tcl`

Altair Monitor-Basic Setup

Altair Accelerator uses a two-layer interface to FlexNet Publisher. There is a daemon `vovresourced` which is started by the NC server whenever there are `vtk_flexlm_monitor` statements in the `resources.tcl` file. `vovresourced` gets license information from `licmcon` via HTTP, and uses it to maintain NC's License: `resources`.

In the case where many NC queues or FlowTracer projects are running, it greatly reduces the load on FlexNet Publisher and improves the consistency of the license information to run Monitor or Monitor-basic. Additionally, Monitor offers a browser interface which shows licenses in use and license utilization information. Altair Accelerator recommends that you run this whenever you need to monitor FlexNet Publisher licenses, even in cases where you only have one server running Accelerator.

Configure and Manage Monitor-basic

Both Monitor-basic and full Monitor are managed by the `lmmgr` command. It is essential that you use this instead of other ways to start the LM vovserver, because it also takes care of any DB schema updates, auxiliary daemon and startup changes needed by newer versions.

You use the configuration file of `vovflexlmd` in Monitor-basic to specify which FlexNet Publisher licenses you want the daemon to monitor. You do not need to monitor all licenses. In this exercise, we will monitor one. In most cases it is easier to use the browser UI to configure which license servers are monitored.

1. Change to the working directory of the daemon and see whether it is already running.
2. If you see an `info.tcl` file, then check whether the process named there exists on that system.

```
% cd $VOVDIR/local/vovflexlmd; ls
```

Example `info.tcl` file:

```
# This file is created automatically by vovlmd: DO NOT EDIT
# If you touch this file, the process 17400 will exit.
set host      "jaguar"
set pid       17400
set port      5555
set cwd       "/remote/proj9/cadmgr/licmon/licmon.swd/vovlmd"
set version   "2.0"
set timestamp 1441146545; # Tue Sep 01 15:29:05 PDT 2019
```



Note: If this directory already contains a file named `config.tcl`, your site may already be running. In this case, check for the file `info.tcl`. This file will contain the host name, process ID, and port of the daemon. You can verify that it is running by pointing your web browser to the URL `http://host:port` using the port and host in the info file.

Even if the daemon is not running, we will use our own subdirectory to run our instance Monitor-basic, rather than use the default directory.

3. Create a directory to run your own Monitor-basic instance.

```
% mkdir ~/ncadmin/licmon
% cd ~/ncadmin/licmon
```

4. Start Monitor-basic, picking a port which is different from the regular one, if you are running on the Accelerator server host. The default TCP/IP port number is 5555. You can find the ports in use by `vovproject list -a -l`. For this example, we use port 5575.

```
% cd ~/ncadmin/licmon
% lmmgr start -name licmon<user> -dir . -port 5575
```

5. Prepare a configuration file with a text editor, which names one of the FlexNet Publisher license files at your site.

```
% vovproject enable licmon<user>
% cd `vovserverdir -p vovlmd`
% vi config.tcl
```

6. Edit the default configuration file `config.tcl`, and enter at least one license file to monitor, using the `add_LM_LICENSE_FILE` procedure. The license may be in any format acceptable as a `-c` parameter to `lmstat`, that is, a full pathname or in `port@host` notation. It is better to use `port@host` than file paths, so that NFS is not involved. Example `config.tcl` file.

```
# config.tcl -- vovflexlmd FLEXlm configuration
add_LM_LICENSE_FILE -tag CDNS /remote/vendors/cadence/license/license.dat
add_LM_LICENSE_FILE -tag SNPS /remote/vendors/synopsys/license/pluto.dat
add_LM_LICENSE_FILE -tag SUNW 1726@saturn
```

7. Verify that LM-basic is running and supplying license information. Examine the `info.tcl` file created by the `vovlmd` daemon, and point your web browser to the URL as described above.
8. Stop the LM-Basic `vovserver`.

Most of the Accelerator daemons may be stopped by touching the `info` file that they create. When it starts, the daemon records the timestamp of the file. Each cycle, the daemon examines the timestamp, and exits if it has changed.

For Monitor-basic, the auxiliary daemons like `vovlmd` will stop when the Monitor-basic `vovserver` stops.

vtk_flexlm_monitor Procedures

Your Accelerator setup will get license in-use info from your Monitor or Monitor-basic setup via HTTP on Monitor's `/raw` interface. The default `resources.tcl` file for Accelerator includes the `vtk_flexlm_monitor_all` procedure. This converts every feature monitored by Monitor into an Accelerator resource. If the resource name is specified, then this name is the actual resource name used. If the resource name is not specified, the name defaults to `License:<feature>`.

If you do not wish to have a large number of resources, you can comment out the above and put calls to the `vtk_flexlm_monitor` procedure in your `resources.tcl` file to specify which FlexNet Publisher features Accelerator should monitor.

Resource Throttling

In some cases, you may wish to restrict the number of a feature which are consumed by Accelerator jobs, to leave some for 'fast' or interactive use.

To accomplish this, you can use an auxiliary resource called `throttle_<resource>` to restrict the number of Accelerator jobs. For example, you have 15 licenses of `hsm`, and want to leave 1 free for jobs that bypass the Accelerator system. You would create a resource 'throttle_hsm' with a count of 14, by adding statements like this. Map `hsm_license` to `throttle_hsm`, so that when the throttle resource is exhausted, jobs that need `hsm` will queue. Often such resources are named `Limit:<something>`.

```
vtk_flexlm_monitor hsm hsm_license throttle_hsm
vtk_resourcemap_set throttle_hsm 14
```

Upgrade Accelerator

There are two types of updates for Accelerator, patches and new versions. Patches replace specific files in an installation and save an archive of the changed files so you can revert. A new release is a complete set of files, usually installed in a directory that is a sibling of your current version.

The simplest method to change to a different server is to use the following steps (shutdown and restart on new version):

- Install new software
- Notify users that Accelerator will be shutdown
- At an idle time, shut down the Accelerator server with `ncmgr stop`
- Change startup files (`.vovrc`) to refer to the new software installation
- Start the server, running the new version, with `ncmgr start`
- Notify users that Accelerator is available again

In some cases, for example, it may be impractical to shut down the Accelerator server for even a short time:

- you have a very busy installation, and Accelerator is never idle
- you have jobs running and you do not want to lose them
- you want to keep the order of jobs in the queue. In such cases you can use one of the following methods.
 1. Switch `vovserver` and `vovtaskers` separately, using `ncmgr stop -freeze`.
 2. Start a new Accelerator queue on new version, sending new jobs to it, shutting down the current Accelerator queue after all jobs have been retired.

Here are the steps for the `ncmgr stop -freeze` method.

1. From a shell with your current Altair Accelerator version, stop `vovserver` with `ncmgr stop -freeze`. This instructs `vovtaskers` with jobs to wait for a new `vovserver` and reconnect to it.
2. `vovtaskers` with no jobs should exit right away, and ones with jobs will enter the `SUSP` state. Their names will change to indicate they are stopping (and to permit new `vovtaskers` to start with the regular names).
3. From a shell with the new Altair Accelerator version, start the `vovserver` using `ncmgr start`.
4. Check on the browser UI Admin page that `vovserver` is running the expected version.

Here are the steps for the overlapping queues method.

1. Create a new queue, running the new server software
2. Direct `nc run` to send new jobs to the new queue
3. After some time, all jobs in the old queue have completed
4. The old queue may be now be shut down

Overlapping Queues

Please read these steps to the end and understand them before doing the procedure. This procedure is preliminary, and may need to be adapted to your Accelerator configuration.

1. Install and verify the new software version. The `vovcheck` command may be helpful.

2. Start a new queue, for example with name `vnc2`.

```
% ncmgr start -queue vnc2
```

3. Copy configuration files (`policy.tcl`, `resources.tcl`, `taskers.tcl` etc.) from the configuration directory of the old queue to that of the new queue in such a way that the two queues are functionally the same.
4. Get the new queue ready.

```
% ncmgr reset -queue vnc2 -full
```

5. In the `setup.tcl` file of the new queue, set the environment variables `NC_OLDQUEUE` and `NC_OLDVERSION` to point to the old queue.

```
# This is a fragment of the setup.tcl file for the new queue
# USE THESE FOR MIGRATION FROM AN OLD QUEUE.
setenv NC_OLDQUEUE
vnc setenv NC_OLDVERSION 5.4.7
```



Note: This goes in the `setup.tcl` file so that these environment variables will always be set in the context of `vnc2`.

6. Test the new queue. By setting the environment variable `NC_QUEUE` to `vnc2`, the Accelerator commands will use that as the default queue.

```
% setenv NC_QUEUE "vnc2"
% nc run sleep 10
% nc list
% nc mon
```



Note: You may use the `-queue` option as an alternative to setting `NC_QUEUE`. For example, you might use

```
% nc -queue vnc list -a      # Show all jobs in old queue.
% nc -queue vnc2 run sleep 10 # Submit a sleep job to new queue.
% nc -queue vnc2 list       # Show your jobs (only) in new
                             queue.
```

7. Once you are satisfied that the second queue is ready, make it the default queue. The default queue is always named `vnc`, and is where the newly-submitted jobs will be placed.

```
% cd $VOVDIR/local/vncConfig
% mv vnc.tcl vnc1.tcl; ln -s vnc2.tcl vnc.tcl
```



Note: In the example above, `vnc.tcl` is moved to `vnc1.tcl` to preserve its contents, in case you want to reactivate the original queue. The two commands are on the same line, separated by a semicolon, so that the file `vnc.tcl` will be unavailable for the shortest possible time. Check carefully!

Now all the newly-submitted jobs will go to `vnc2`, while commands such as `nc info`, `nc stop`, `nc forget`, etc. will be run on both queues.

8. Error messages are filtered out, because a job ID which is valid in the first queue will not be valid in the second, and conversely. To see the results with messages, query the queues separately by using the `-queue` option of Accelerator commands.
9. When all jobs in the old queue have been retired, you can shut down the old queue.

Use Altair Accelerator's REST API to Submit and List Jobs

This chapter covers the following:

- [Key REST Concepts](#) (p. 80)
- [Get Ready for REST](#) (p. 82)
- [Example Application: REST 101](#) (p. 83)
- [Launch Jobs with Non-default Options](#) (p. 87)
- [The vov_rest_v3.py Python Library Module](#) (p. 88)
- [REST Request Detailed Documentation](#) (p. 90)
- [Issuing REST Requests from the Swagger Web UI](#) (p. 91)
- [Advanced REST Usage](#) (p. 93)
- [JWT Token Allocation](#) (p. 96)

Overview

Altair Accelerator supports a Representational State Transfer (REST) API that enables a developer to submit, monitor, and control batch jobs as well as query the batch scheduler queues, jobs, and hosts in useful ways. A REST API is based on a URL name space targeted by HTTP operations like POST and GET to submit jobs or make queries. REST APIs are popular ways for developers to write portable applications, both UI and command line, that can interface to the Accelerator queue over secure HTTPS connections. This tutorial will introduce key concepts and will also provide working example Python programs that interface to Accelerator via REST.

Developers use a variety of languages in web clients when interfacing to a REST API, including Python, PHP, Java, Node.js and others. This tutorial will focus on the use of Python in example code.

For more specific information, see [REST API](#) in the *VOV Subsystem Reference Guide*.

Also in this Tutorial

Key REST Concepts

The components of a REST operation are the following:

- Resource Path
- HTTP Verb
- Body
- Header

The URLs in the application's REST URL name space provide the targets for the read (GET), write (POST), or update (PATCH) HTTP request types. The parameters of the REST operations always include a URL and an access token that authorizes access to the services. The following sections describe more about these concepts.

Resource Path

The resource path describes the object to be acted upon. It is in the form of a URL with the following structure:

```
BASE_URL + "/api/" + VERSION + "/" + REST
```

BASE_URL is the initial part of the URL for the Accelerator queue to access. For example, if the Accelerator NC_QUEUE environment variable was set to My_Test_Queue, then use the `nc cmd vovbrowser` command to obtain a BASE_URL.

The VERSION is the REST API version, currently v3.

REST is the remaining part of the URL that applies to the specific types of objects you will be referencing.

The resource path might look something like this when you are referencing the version information about the Accelerator queue scheduling server known as vovserver.

```
https://server.domain.myco.com:6330/api/v3/project/1/version
```

HTTP Verb

The verb describes the action to take regarding the resource.

POST	Creates a resource
GET	Retrieves one or more resources
PUT	Updates or controls a resource
PATCH	Updates a resource
DELETE	Deletes a resource

Body

The body of a REST request defines parameters and options that apply to the action being taken. A POST or PUT request has a body. A GET or DELETE request has no body. A Python dictionary data structure is the usual format for the body, and it consists of a set of keywords with associated values.

Header

The header contains metadata about the message being sent to the server which includes, importantly, an *access token*. REST HTTP requests are initiated across the network from a node other than the one on which the Accelerator server is running. The access token provides the assurance that the sender has permission to request the REST action on the server.

An authentication request type is available via a POST that supplies a username and password to the server, to which the server responds with the access token. When a REST request comes in later with this access token, the server can quickly and reliably extract the user identity and permissions as a prelude to processing the request. Accelerator REST utilizes a style of access token known as JSON Web Tokens (JWTs).

Get Ready for REST

REST API usage prerequisites are described in this section. These prerequisites will enable REST v3 applications to run with Accelerator. Some of the prerequisites apply to the Accelerator queue configuration and others apply to the host where the REST application will run, also called the "submit host".

Prerequisites for the Accelerator queue:

- The web port must be configured. In version 2021.2.0 through 2022.1.1, the web provider must be set to "internal". See the `-webport` and `-webprovider` options in the `-h` help screen documentation for VOV project start commands like `ncmgr start` and `lmmgr start`.
- **Optional:** SSL/TLS should be enabled for security reasons. Since passwords are passed with HTTP authorization requests to `vovserver`, the security of the connection is important. Enable SSL/TLS for the NC queue as follows:

1. Add this line to `SWD/policy.tcl`:

```
set config(ssl.enable) 1
```

2. If REST requests will be sent from Python scripts running on CentOS 7 or earlier, TLS 1.2 will be used by the REST application you are writing. Configure `vovserver` to accept TLS 1.2 protocol by adding the following line to `SWD/policy.tcl`:

```
set config(http.minSSLVersion) "TLSv1.2"
```

3. **Optional:** Append a line to `SWD/setup.tcl` that sets `VOV_HOST_HTTP_NAME` to the fully qualified host name (FQHN) where `vovserver` is running. The FQHN is the output of `hostname -f`.

```
setenv VOV_HOST_HTTP_NAME FQHN
```

4. **Optional:** For a full HTTPS security, a CA-signed domain-wide SSL certificate is installed in `$VOVDIR/local/ssl` or a host-specific CA-signed SSL certificate is installed in `SWD/config/ssl`. This will allow your REST network traffic to be fully secured. See [Advanced REST Usage](#) for more about this.
5. Reread and activate the changed server configuration parameters via `vovproject reread` or similar commands.

Prerequisites for the submit host:

- The submit host must have network access to the server running the NC queue.
- Python version 3 or higher is required to use the `vov_rest_v3.py` REST access library module.
- The Python "requests" package must be installed.
- Copy `vov_rest_v3.py` from `$VOVDIR/scripts/python/vov_rest_v3.py` to the directory on the submit host where your Python application resides.

Example Application: REST 101

Programs that use the Accelerator REST API can do so in two ways. The preferred approach is to utilize the Python library module "vov_rest_v3", which is provided with Accelerator software packages in the file `vov_rest_v3.py`. This Python module simplifies coding by hiding details of the HTTP operations that interface directly to the low level REST API. The `vov_rest_v3` module requires Python version 3 or higher.

For more advanced users, a later section of this guide shows how to interface directly to the REST API. Direct use can give you more control over the following:

- Management of JWT access tokens
- Policy of your application regarding insecure HTTPS configurations
- Control over the use of connection "keep-alive"

Direct REST API interface will be covered later. This approach works with Python version 1 and higher.

To get a quick start using REST, examine a simple "REST 101" example Python program that queries REST to return the name of an NC queue, shown below. The lines of the program are numbered and annotated with explanations.

```
1  #!/usr/bin/python3
2  #
3  # nc_rest_101.py
4  #
5  import os, sys, getpass, json
6  import vov_rest_v3
7
8  url = os.environ['NC_URL']
9
10 vrest = vov_rest_v3.VOVRestV3()
11 vrest.authorize(url, getpass.getpass('Password:'))
12 r = vrest.submitRequest("GET", url + "/api/v3/project/1/project")
13
14 print ( json.dumps(json.loads(r), indent=2) )
```

Here is a line-by-line guide to the above program.

- | | |
|----------------|--|
| Line 1 | This Python program is compatible with Python 3. |
| Line 6 | Import the <code>vov_rest_v3</code> Python module that comes with the Accelerator product. Copy it locally from <code>\$VOVDIR</code> before running the Python program. |
| Line 8 | Pull the URL for the NC queue out of an environment variable. The URL is what <code>nc cmd vovbrowser</code> shows. |
| Line 10 | Allocate <code>vrest</code> , an instantiation of the <code>VOVRestV3</code> Python class. |
| Line 11 | The <code>authorize()</code> method function authenticates using the password and allocates a JWT token stored in the object. This example obtains your user password by interactive prompt. |



Note: This example uses username/password authentication. To use key-based authentication, see .

Line 12 The HTTP get request is sent, returning a string containing the HTTP response data in JSON format.

Line 14 The HTTP response string is parsed into JSON and pretty-printed.

Here is a terminal session that shows how to run the program.

```
% echo $NC_QUEUEE
My_NC_QUEUEE
% export NC_URL=`nc cmd vovbrowser`
% echo $NC_URL
http://myhost:6330
% cp $VOVDIR/scripts/python/vov_rest_v3.py .
% python3 nc_rest_101.py
Password:
{
  "startrow": 1,
  "endrow": 1,
  "query": "SELECT project FROM 1",
  "errmsg": "",
  "columns": [
    {
      "col": 0,
      "field": "project"
    }
  ],
  "rows": [
    [
      "My_NC_QUEUEE"
    ]
  ]
}
```

Example Applications: Job Submit and List

To demonstrate the utility of the REST API, here are two simple Python programs that imitate the function of the `nc run` and `nc list` commands that are familiar CLI tools from the Accelerator product. The source code for these tools, named `nc_run.py` and `nc_list.py`, is found on the following pages.

These REST programs only require that the `NC_URL` environment variable be set to the Accelerator queue URL, as returned by `nc cmd vovbrowser`. Here is a terminal image that demonstrates the simple REST tools.

```
% cp $VOVDIR/scripts/python/vov_rest_v3.py .
% export NC_URL=`nc cmd vovbrowser`
% ./nc_run.py sleep 33
Password:
New job is 45080
% ./nc_run.py sleep 66
Password:
New job is 45083
% ./nc_list.py
Password:
```

ID	STATUSNC	PRIORITYP	HOST	COMMAND
000045080	Running	normal	myhost	vw sleep 33 > JOBL0G.185633
000045083	Running	normal	myhost	vw sleep 66 > JOBL0G.903723

nc_run.py

```
#!/usr/bin/python3
#
# nc_run.py
#
# Usage
#
#     export NC_URL=<URL FOR NC QUEUE>
#     ./nc_run.py <command> [args]
#
import os, sys, random, json, getpass
import vov_rest_v3

def getMyPassword():
    return getpass.getpass('Password:')

# Main body
nc_url = os.environ["NC_URL"]
scheme = nc_url.split(":")[0]
hostport = nc_url.split("/")[2]
url = "{0}://{1}".format(scheme, hostport)
command = " ".join(sys.argv[1:])

vrest = vov_rest_v3.VOVRestV3()
vrest.authorize(url, getMyPassword())

# Job attributes - required
VOV_JOB_DESC = {
    "command" : command,
    "logfile" : "JOBL0G." + str(random.randint(100000,999999)),
    "rundir" : os.getcwd(),
    "env" : "BASE",
}

# Job attributes - optional / User specified
VOV_JOB_DESC.update( {
    "priority,sched" : 4,
} )
r = vrest.submitRequest("POST", url + "/api/v3/jobs", jsonData=VOV_JOB_DESC)
print ("New job is %s" % json.loads(r)["jobid"])
```

nc_list.py

```
#!/usr/bin/python3
#
# nc_list.py
#
# Usage
#
#     export NC_URL=<URL FOR NC QUEUE>
#     ./nc_list.py
#
import os, sys, json, getpass
```

```
import vov_rest_v3

def getMyPassword():
    return getpass.getpass('Password:')

def listJob(vr, url):
    query = ( url + '/api/v3/query'
              + '?select=id,statusnc,PRIORITYPP,host,command'
              + '&from=System:User:' + os.environ['USER'] )
    response = vr.submitRequest("GET", query)
    return response

def prettyPrint( text ):
    dd = json.loads(text)
    for ii in range(0, len(dd['columns']) ) :
        sys.stdout.write("%9.9s " % dd['columns'][ii]['field'])
    sys.stdout.write("\n")
    if ('rows' not in dd):
        return
    for rr in range (0, len(dd['rows']) ) :
        row = dd['rows'][rr]
        for ii in range(0, len(dd['columns']) ) :
            if (ii < len(dd['columns'])-1):
                sys.stdout.write("%9.9s " % str(row[ii]))
            else:
                sys.stdout.write("%10.30s" % str(row[ii]))
        sys.stdout.write("\n")

#
# Main body
#
nc_url = os.environ['NC_URL']
scheme = nc_url.split(":")[0]
hostport = nc_url.split("/")[2]
url = "{0}://{1}".format(scheme, hostport)

vrest = vov_rest_v3.VOVRestV3()
vrest.authorize(url, getMyPassword())
json_text = listJob(vrest, url)
prettyPrint(json_text)
```

Launch Jobs with Non-default Options

The `nc_run.py` example program shown earlier can submit an Accelerator job with no customizations. In reality, users often need to specify non-default properties for the jobs they submit. Examples are resources, slot counts, auto-kill times, and numerous other options that can be seen on the `nc run -h` CLI command help information and in the documentation with the *Altair Accelerator Administrator Guide*.

To view available job launch options, find the `VOV_JOB_DESC` table of options in [Define Jobclasses](#) of the *Altair Accelerator Administrator Guide*. Here is an excerpt of that table:

Field in Array	Description
autokill	Set the autokill flag (option -kill)
check,directory	Set it to 0 to disable checking of canonicalization of current directory (option -D)
env	Environment of the job (option -e). Set this to "" or to DEFAULT to force the use of an environment snapshot.
force	Force the job to be rescheduled (option -F)
group	Group the job belongs to (options -g and -G)
inputs	List of input files (dependencies) (option -i)
priority, default	Default priority (NOT USED)

Although the above document has good descriptions of the fields/keys that can be provided to job launch requests, some of those fields are usable in Accelerator jobclass definitions but not in REST job requests. To see the precise list of fields/keys that are supported in REST requests, see the Swagger docs described in the [REST Request Detailed Documentation](#) section later in this tutorial.

You can modify the `nc_run.py` script previously shown (or the `nc_run2.py` script in an upcoming section) with additional options specified as key-value pairs in the `VOV_JOB_DESC` Python dictionary. For example, here is the modification to `nc_run.py` that specifies an auto-kill time of 60 seconds. The Python dictionary “update” method function provides a convenient way to do that.

```
VOV_JOB_DESC.update( {  
    "autokill"      : 60,  
} )  
  
vrest.submitRequest("POST", resturl, jsonData=VOV_JOB_DESC)
```

The `vov_rest_v3.py` Python Library Module

The Accelerator software package provides a Python library module called `vov_rest_v3.py` to make REST API usage from Python more convenient. The module implements a `VOVRestV3` Python class with member functions described in the following text box. These functions hide the details associated with authenticating, session handling, and error handling. More example programs follow that utilize this convenient Python library layer.

VOVRestV3 Python Class Description

The `VOVRestV3` Python class provides an interface to the Accelerator v3 REST API.

Location

```
$VOVDIR/./common/scripts/python
import vov_rest_v3
```

Member Functions

authorize (url, username='', password='')

The vovserver scheduling server authenticates a user for a `VOVRestV3` and obtains a validating access token behind the scenes, storing it in the `VOVRestV3` object.

Positional Arguments `url` – (string) the URL as returned by `nc cmd vovbrowser`

`password` -- (string) the current user password

Keyword Arguments `username` – (string) user name, usually the same as the Linux user name known to vovserver. If this argument is not provided it defaults to the current user identified by `$USER`

Return Value None. Raises exceptions upon failure.

submitRequest (method, url, queryParams={}, jsonData={})

Submits a REST/ HTTP request to the server.

Positional Arguments `method` – (string) one of: “GET”, “POST”, “PUT”, “DELETE”, “PATCH”.

`url` – (string) the URL as returned by `nc cmd vovbrowser -url RESTPATH`.

Examples of `RESTPATH` are: `/api/v3/jobs` or `/api/v3/projects/1`.

Keyword Arguments `queryParams` – (dictionary) keywords and values in string format for GET requests

`jsonData` – (dictionary) keywords and values in string format for POST requests

Return Value (string) The HTTP request response text.

getJWT()

Retrieve the JSON Web Token (JWT) for the object.

Return Value: (string) the JWT string

setJWT(token)

Replace the JSON Web Token (JWT) for the object.

Positional Arguments token – (string) the replacement JWT to be remembered in the object and used for subsequent REST requests

Return Value None.

REST Request Detailed Documentation

The Accelerator REST v3 API interface is described in detail under the Swagger documentation that comes with the Accelerator software. To browse the Swagger REST documentation, browse to the URL displayed as the output of this command:

```
nc cmd vovbrowser -url /html/vovrest.html
```

The information at this page will help you construct a valid REST request.

For example, browse on the Swagger documentation page to the “jobs” object. You will see that a POST request is used, and the REST URL segment is “/api/v3/jobs”.

jobs		▼
POST	/jobs Create job	🔒
PUT	/jobs/{id} Job Control	🔒
PATCH	/jobs/{id} Modify job	🔒
DELETE	/jobs/{id} Removes job from server memory	🔒
GET	/jobs/{id} Query all fields on a single job.	🔒
GET	/jobs/{id}/{field} Query single field on a single job.	🔒

Figure 14: REST Object-Verb Reference

Click on the POST, PUT, PATCH, DELETE, or GET buttons for more details about the required parameters and supported dictionary keys and values for each request type. This gives you what you need to know to construct a complete and working job submit request via Python or curl.

Issuing REST Requests from the Swagger Web UI

The Swagger Rest API page provides a learning interface that helps you formulate and execute well-formed REST requests to the associated NC queue. The web UI page shows a construction of the REST request and response text in curl command and JSON language formats.

Here are the steps to formulate and execute a REST request:

1. Access the VOV REST API page in a browser. The URL is shown by the command:

```
nc cmd vovbrowser -url /html/vovrest.html
```

2. If SSL/TLS is enabled for the NC queue, then select the https URL in the "Server" drop-down menu on the left side of the page near the top.
3. Scroll down to the Object-Verb section for the desired REST request.
4. Click on **Try it Out** on the right.
This activates a web form on the page to accept your specified parameters for a REST request before executing it.
5. Edit the parameters for your REST request in the web form.

Example A

If you are selecting a GET operation on a job object, the job id and sometimes a job object field name are required. Enter these items in the web form.

Example B

If you are selecting a POST operation on a job object to submit a job to Accelerator, the parameters are more extensive. Edit the job creation parameters box. A template window in the web form illustrates the format, which is consistent with a Python dictionary syntax with keyword/value pairs. Replace the template with your desired job creation specifications.

Most of the keyword/value pairs in the template for job creation are optional, but a few are required. A simple and minimal working example job creation parameter list follows. To specify other properties of a job, choose some additional keywords from the template text that appears pre-populated in this sub-window.

```
{
    "command" : "sleep 60",
    "logfile" : "JOBLOG.01",
    "rundir"  : "/tmp",
    "env"     : "BASE"
}
```

6. Click the blue **Execute** button.
This sends the REST request.
7. Scroll down to see the Server Response. A code of 200 - 299 indicates success.
8. Troubleshoot if the request failed. Common errors and remedies include:
 1. If error text is "Bad Request", then check that the right URL is selected in the Servers drop-down menu at the upper left.
 2. If an authentication error is seen, logout of the web UI and then log in again so the Swagger page gets a fresh JWT access token.
 3. If an error with "Error: Bad Request" is seen, check the specified text in the Job Control Parameters sub-window. A common mistake is to add an invalid comma after the last keyword/value pair.

4. A server error is returned if you specify a job logfile that is the same as another job in the system. Change the logfile name and retry the request.

Advanced REST Usage

This section will cover some more advanced REST API topics: management of JWT access tokens, HTTPS secure and insecure connections, and connection keep-alive. The example application, `nc_info.py`, imitates the familiar Accelerator CLI command `nc info`, which returns information about a job. This application will interface to the REST API directly, instead of using the `vov_rest_v3` module interface layer.

This REST application requires the `NC_URL` environment variable to be set to the Accelerator queue URL, as returned by `nc cmd vovbrowser`. Also, this Python program needs JWT-handling module `getToken.py` from the next section in this guide. Create that python file before running `nc_info.py`.

Here is a shell session that shows how to run `nc_info.py`. In this example, two jobs are started, and an invocation of `nc_info.py` will query and display information about the two running NC jobs.

```
% nc run -v 2 sleep 123
Job <000001138> is submitted
% nc run -v 2 sleep 456
Job <000001143> is submitted
% export NC_URL=`nc cmd vovbrowser`
% echo I will need `ls getToken.py`
I will need getToken.py
% ./nc_info.py 000001138 000001143
Password:
Job                000001138
User,Group         user99,/time/users.user99
Command            vw sleep 123 > vnc_logs/20211012/132628.122875
Status             Running
  Host             myhost
  Duration         31s

Job                000001143
User,Group         user99,/time/users.user99
Command            vw sleep 456 > vnc_logs/20211012/132633.122899
Status             Running
  Host             myhost
  Duration         26s
```

nc_info.py

```
#!/usr/bin/python
#
# nc_info.py
#
# Usage
#
#     export NC_URL=<URL FOR NC QUEUE>
#     ./nc_info.py JOB_ID [JOB_ID ...]
#

import os, sys, json, requests
from getToken import getJWT

def getMyPassword():
    import getpass
    return getpass.getpass('Password:')

def infoPrint( text ):
    dd = json.loads(text)
    id   = find(dd, 'ID')
    user = find(dd, 'USER')
    group = find(dd, 'GROUP')

    print ("% -16s%s" % ("Job", id) )
    print ("% -16s%s,%s" % ("User,Group", user, group))
    print ("% -16s%s" % ("Command", find(dd, 'COMMAND')))
    print ("% -16s%s" % ("Status", find(dd, 'STATUSNC')))
    print ("% -16s%s" % ("  Host", find(dd, 'HOST')))
    print ("% -16s%s" % ("  Duration", find(dd, 'DURATIONPP')))

def find(jobdump, key):
    for c in range (0, len(jobdump['columns'])):
        if (jobdump['columns'][c]['field'] == key):
            break
    return jobdump['rows'][0][c]

#
# Main body
#
nc_url = os.environ['NC_URL']
scheme = nc_url.split(":")[0]
hostport = nc_url.split("/")[2]
url = "{0}://{1}".format(scheme, hostport)

ss = requests.Session() # use keep-alive
jwt = getJWT(ss, url, os.environ["USER"], getMyPassword())
for arg in range (1,len(sys.argv)):
    jobid = sys.argv[arg]
    query = url + '/api/v3/jobs/' + jobid
    r = ss.get(query, headers={"Authorization": jwt},
    verify=True)
    infoPrint(r.text)
    print ("")
```

JWT Access Tokens

To authorize REST access via the API, REST requests must pass an access token in the request header. In the `nc_info.py` example, the `jwt` variable holds the access token. Access tokens expire about 4 hours after issue. An application that will run for several hours should adopt some strategy to allow for access token expiry. One strategy would be to re-authenticate, using login name and password, prior to any burst of REST activity that will be known to be complete in a few hours or less. Another strategy is to check the request return status and re-authenticate at that time, using the new access token thereafter.

The recommended way to authenticate user name and password to issue an access token is the `VOVRestV3` Python class `authorize()` method function. If you would like direct control over the access token allocation, as in the `nc_info.py` example, see the `getJWT()` function in the `getToken.py` example in the next section.

HTTPS Security Considerations

REST requests and responses are sent over the HTTP communication protocol, and suitable HTTP connections can be configured with three possible security levels, ranging from insecure to very secure:

1. The basic (insecure) way to start the Accelerator queue server and web server is to use the default HTTP connection on the server's webport. The REST API interface always is allowed on HTTP webport connections.
2. An intermediate level of security is possible if the HTTPS protocol is requested by configuring server parameter `ssl.enable` to 1 in `policy.tcl`. In this case, a self-signed SSL certificate will be generated when the Accelerator server starts. If the REST application uses the `VOVRestV3` python module method functions, this type of connection will be supported without warnings being issued. If direct access to the REST API is implemented using the Python request method functions `get()` and `post()`, then communication will be allowed on these connections only if the optional `verify=False` keyword argument is passed to those functions.
3. A very secure HTTPS connection will be established if an SSL certificate that was signed by a trusted Certificate Authority (CA) is added to the Accelerator server configuration. This is the recommended way to configure Accelerator products. If the REST application author would like to require level 3 security, then the direct access to REST via request method functions `get()` and `post()` must be used, and the keyword argument `verify=True` must be specified.

Connection Keep-Alive

The use of HTTP keep-alive, or persistent connection, is an important technique that optimizes applications during times of very frequent REST requests. HTTP keep-alive and the resultant reuse of HTTP connections will occur when using the `VOVRestV3` `submitRequest()` method function or when using a "session" allocated by the `request.Session()` method function.

The `nc_info.py` Python application illustrates the use of keep-alive by the latter method in its main loop across job IDs. Each call to `ss.get()` will reuse the same HTTP connection. If the `ss.get()` call were to be replaced by `request.get()`, the keep-alive feature would not persist an HTTP connection between subsequent requests. In that case, each request would build a new HTTP connection before issuing the request.

JWT Token Allocation

Accelerator REST API v3 uses JSON Web Tokens (JWTs) to implement the access tokens used in REST requests. A JWT access token is allocated by an HTTP POST request by the client that passes a matching Linux username and password along with an implementation-defined REST URL. The server responds by verifying the username and password match and returning the JWT access token to the client.

The `authorize()` method function in the `VOVRest` Python module should be used for most convenient authorization and automatic JWT handling. This method is used in the later examples in this tutorial. The `getToken.py` module that follows shows the low level interface to JWT allocation. This module must be provided with the first few working Python code examples shown in this tutorial.

`getToken.py`

```
# JWT Token utilities

import os, requests

#
# Function getJWT()
#
# Arguments
#
#     url           - A URL for a VOV project
#     user          - user name for authentication
#     password      - password for authentication
#
def getJWT(url, username, password):
    scheme = url.split(":")[0]
    hostport = url.split("/")[2]
    baseUrl = "{0}://{1}".format(scheme, hostport)
    tokenUrl = baseUrl + "/api/v3/token"
    myauth={ 'username' : username , 'password' : password }
    r = requests.post(tokenUrl, data=myauth)
    if ( r.status_code > 300 ):
        print ("JWT Error code %d" % r.status_code)
        print ("           returned status: ", r.json() )
        print ("           error message : %s" % r.json()['error'])
        exit(1)
    token = r.json()
    jwtToken = token['token_type'] + ":" + token['access_token']
    return jwtToken

#
# Function getMyPassword()
#
# This example function simply prompts the user to type the account password.
#
def getMyPassword():
    import getpass
    return getpass.getpass('Password:')
```

The Python code in the `getToken.py` module also contains a placeholder password prompt function `getMyPassword()`. The handling of user passwords and JWT tokens in practice will be up to the REST application developers in accordance with their own best practices for handling and storing security-sensitive information. If the REST application runs for many hours, new

JWT tokens will need to be allocated and authenticated after the previous ones expire. The application needs to provide a way to provide the password each time a JWT token is authenticated. Additional methods for renewing or allocating JWT tokens are being considered for future Accelerator software releases.

This case study provides one example of sophisticated usage of Allocator.

Assume:

- There are 3 physical locations: US, Asia and Europe
- Each location has its own installation of Accelerator
- Each location has its own set of licenses (in our example, we use calibre)

The goal of the case study is to achieve:

- Some licenses to be used locally only
- Some licenses to be shared across all locations
- Each Accelerator can continue to use its own local licenses even if Allocator goes down for any reason (such as, network outage)
- Users do not have to know what `LM_LICENSE_FILE` to set before submitting jobs. The correct `LM_LICENSE_FILE` environment variable is set automatically by Accelerator at run time.

Step 1: Setup Monitor on Each Site

Assume in US site, we have two license servers, namely `27001@earth` and `27002@jupiter`. We want to restrict calibre usage served by `27001@earth` to US local Accelerator queue, and share `27002@jupiter` with the rest of the queues.

Since they serve the same feature, you need to tag the FlexNet Publisher daemon when you configure Monitor. For example, in your `config.tcl` for `vovlmd`:

```
#
# Example of vovlmd config.tcl
#
add_LM_LICENSE_FILE 27001@earth -tag US_LOCAL
add_LM_LICENSE_FILE 27002@jupiter -tag US_WAN
```

You can do a similar setup for all other locations.

Step 2: Setup resources.tcl to Monitor the License Features of Interest

Assume the feature name is `calibre`. In the `vnc.swd/resources.tcl` file of US Accelerator queue, you could add:

```
#
# Fragment of resources.tcl for the US site
#
# Use LOCAL:calibre as Accelerator resource name for calibre from 27001@earth.
vtk_flexlm_monitor US_LOCAL/calibre LOCAL:calibre

# Use WAN:calibre as Accelerator resource name for calibre from 27002@jupiter.
vtk_flexlm_monitor US_WAN/calibre WAN:calibre

# When user submits a job requesting License:calibre, use either LOCAL or WAN
# resource, with LOCAL preferred.
```

```
vtk_resourcemap_set License:calibre UNLIMITED "LOCAL:calibre | WAN:calibre"
```

After changing `resources.tcl`, run `ncmgr reset` on the command line and use the resources monitor (`nc_mon`) to verify that you get the correct number of resources for both `LOCAL:calibre` and `WAN:calibre` and you have `License:calibre` correctly map to `"LOCAL:calibre | WAN:calibre"`.

The setup enables US Accelerator to use the both LOCAL and WAN calibre licenses when calibre is not managed by Allocator.

Repeat the same setup for the other sites as well.

Step 3: Setup Allocator to Manage Calibre Licenses

First we want to add all the Monitors that serve all WANable calibre licenses to Allocator.

```
#  
# Segment of vovlad/config.tcl  
#  
# Monitor for US  
LA::AddLicenseMonitor jupiter:5555  
  
# Monitor for Asia  
LA::AddLicenseMonitor tiger:5555  
  
# Monitor for Europe  
LA::AddLicenseMonitor queen:5555
```

Since the Allocator daemon checks the config file at every cycle, and reads it if it has changed, you may have to wait 1 or 2 minutes to see the changes reflected in the Allocator page.

Then you need to add the resource to be managed by Allocator.

```
#  
# Segment of vovlad/config.tcl  
#  
#  
# Manage resource 'WAN:calibre', which is derived from  
# the specified list of tags for the FlexNet Publisher feature 'calibre'  
#  
LA::DefineResourceGroup WAN:calibre {  
  LA::AddResource WAN:us_calibre FlexLM/US_WAN/calibre ""  
  LA::AddResource WAN:jp_calibre FlexLM/ASIA_WAN/calibre ""  
  LA::AddResource WAN:eu_calibre FlexLM/EUROPE_WAN/calibre ""  
}
```

In the above example, we assume Accelerator is running on `host_us`, `host_asia`, and `host_europe` in US, Asia and Europe, respectively; and their Monitor has the corresponding tags.

We also need to use the `-tags` option to specify what tags are involved in Allocator. That is because, for example, we configured both `US_WAN` and `US_LOCAL` tags in the same Monitor. When Allocator connects to this Monitor server, it needs to tell the difference between wannable and non-wannable licenses to know which license features should be managed by Allocator.

We use `WAN:calibre` as the Accelerator resource name for calibre, which is the same name that we use in `resources.tcl`. By doing this, the resource `WAN:calibre` becomes managed by Allocator rather than `vovresourced` since Allocator has a higher rank than `vovresourced`. When Allocator goes down, the `WAN:calibre` resource set by Allocator will expire automatically after a short while (about 1 minute), which enables `vovresourced` to become the owner of this resource once again.

You also need to define the Accelerator sites and assign relative weights on the resource for each site. For example, the following configuration allows 5:3:2 allocation of WAN:calibre resource among US, ASIA and EUROPE queues:

```
#
# Segment of vovlad/config.tcl
#

LA::AddSite vnc@host_us US {
    WAN:calibre 50
} -port 6271

LA::AddSite vnc@host_asia ASIA {
    WAN:calibre 30
}

LA::AddSite vnc@host_europe EU {
    WAN:calibre 20
}
```

After this setup, check the Allocator CGI page again. You should see the total number of calibre licenses from all sites.

Step 4: Setup Jobclasses for LM_LICENSE_FILE environment variable

Check the [Jobclass](#) documentation for details on how to setup job classes.

To set the LM_LICENSE_FILE environment variable for jobs at runtime, you need to set the [VOV_LM_VARNAME](#)s variable.

```
#
# Example of jobclass/calibre.tcl
#

set classDescription      "Calibre jobs"
set VOV_JOB_DESC(resources) "License:calibre"
set VOV_JOB_DESC(env)     "BASE+CALIBRE+D(VOV_LM_VARNAME=LM_LICENSE_FILE)"
```

Then when you submit jobs, you can use `-C calibre` option. After you submit job, you can use `nc info -l <jobId>` to look at the log file of the job to make sure it does set the correct LM_LICENSE_FILE environment variable.

This chapter covers the following:

- [EDA Demo Part 1: Run the Demo](#) (p. 102)
- [EDA Demo Part 2: Dissect the Demo](#) (p. 110)

The purpose of this tutorial is to illustrate how FlowTracer can be used in the context of a complex chip design. First we guide you through a demonstration, then we analyze the components in detail.

Our assumptions are:

- The chip we are designing is called "gigalite"
- We are using CVS for revision control
- We are emphasizing the browser interface as opposed to the programmable Command Line Interface (CLI) or the dynamic Graphical User Interface (GUI)
- We expect multiple designers to be involved in the design. Each designer has a private workspace. All information exchange among the designers goes through the CVS vault.
- The design flow is based upon a set of (fictitious) tools with name begins with "cdt" (Chip Design Tool). For example, to perform the synthesis of a block called "mmu" we will invoke the command:

```
% cdt_synth mmu
```

- The structure of the chip is described in a file called `$VOVDIR/eda/EDADEMO/chipStruct.tcl`

EDA Demo Part 1: Run the Demo

Setup

You should have already setup your account using `vovsetupuser`.

1. Run `vovcheck` to ensure that there are no problems with your installation:

```
% vovcheck
```

2. Set up the environment `BASE` so that you have `cv`s in your path:

- a) Switch to the environment `BASE`.

```
% ves BASE
```

- b) Check to ensure that `CVS` is in the path.

```
% cvs
Usage: cvs [cvs-options] command [command-options-and-arguments]
...
```

- c) If `CVS` is not available, get the package as follows. You will need to have `sudo` or root access on your Linux box.

```
% sudo yum install cvs
```

Assuming your Linux distribution is RedHat release:

```
% cat /etc/redhat-release
```

CentOS Linux release 7.7.1908 (Core).

If your Linux distribution is another version, please consult your manual on how to access the tool.

3. If `CVS` is not available, access the package as shown below. You need to have `sudo` or root access on your Linux box to complete this step.

```
% sudo yum install cvs
```

Assuming your Linux distribution is RedHat release (CentOS Linux release 7.7.1908 (Core)).

```
% cat /etc/redhat-release
```

If your Linux distro is something else, then please consult your manual as how to get the tool.

Start the Project

1. Create a directory to run the demo.

```
% mkdir ~/EDADEMO
```

```
% cd ~/EDADEMO
```

2. Create a directory for the project administration and another one to do hold the design data.

```
% mkdir ftadmin  
% mkdir work
```

3. Create and start the project called edademo. We want to do this in the environment BASE to make sure that the vovserver has access to cvs:

```
% ves BASE  
% vovproject create -dir ftadmin -type edademo edademo  
% vovproject enable edademo
```

Customize the Project

Because the browser interface for this demo is dependent on a CGI script called `edademo.cgi`, we want to make sure that all the HTML pages generated by the vovserver contain a link back to the CGI pages.

We can do this by controlling the value of the EXTLINKS property.

1. Complete the command below to control the value of the EXTLINKS property.

```
% vovprop set -text 1 EXTLINKS "EDADEMO /cgi/edademo.cgi"
```

2. The tasker list file needs to be customized as well. Open it at `ftadmin/edademo.swd/taskers.tcl`.
3. The default list specifies many taskers on the local host. While this is probably ok for the purposes of the demo, you are encouraged to replace the default list with a list of real machines, as in the following example:

```
# Example of taskers.tcl file.  
vtk_tasker_set_default -resources "@STD@ @RAMTOTAL@"  
  
vtk_tasker_define hyppo -CPUS 4  
vtk_tasker_define rat -CPUS 1  
vtk_tasker_define cat -CPUS 1
```

Check Out the Data

1. Make sure you are in the context of the project edademo using `vovproject enable` and switch to the environment `BASE+EDADEMO` using `ves`.

```
% ves BASE+EDADEMO
```

2. Create a workspace for integration and check-out the data from the CVS repository:

```
% export CVSROOT=$VOVDIR/eda/EDADEMO/cvsVault  
% cd ~/EDADEMO/work  
% mkdir integration  
% cd integration  
% cvs co gigalite  
% export CVSROOT=$VOVDIR/eda/EDADEMO/cvsVault
```

3. Build the flow for integration. The flow we are using is `$EDADEMO/flows/BlockFlow.tcl`:

```
% vovbuild -f $EDADEMO/flows/BlockFlow.tcl
```

Here are the above steps, using a designer called "Mary":

1. If necessary, setup a correct project context and a proper environment:

```
% vovproject enable edademo  
% ves BASE+EDADEMO
```

2. Create a workspace for the user and check-out the data from the CVS repository:

```
% cd ~/EDADEMO/work  
% mkdir mary  
% cd mary  
% cvs co gigalite
```

3. Here we assume that this designer does not need to work on the entire chip, but rather on just a few units.

```
% mkdir local  
% cp $VOVDIR/eda/EDADEMO/chipStruct.tcl local/chipStruct.tcl  
% vi local/chipStruct.tcl  
  
# Example of a local/chipStruct.tcl file for an individual designer  
set listOfUnits {  
    adder rtl  
    alu    rtl  
    cpu   rtl  
    mmu   rtl  
}
```

4. Build the flow for this user:

```
% vovbuild -f $EDADEMO/flows/BlockFlow.tcl
```

5. Repeat the steps for other users as necessary.

Start the Browser Interface

1. Use `vovbrowser` to get the URL of the project

```
% vovbrowser  
http://your_computer_name:6407
```

Here you can see a few examples of what to expect when using the browser interface. The first picture shows the summary for all workspaces in the project, in this case 'integration' and 'mary'. The second image shows the report for the workspace for user 'mary'.

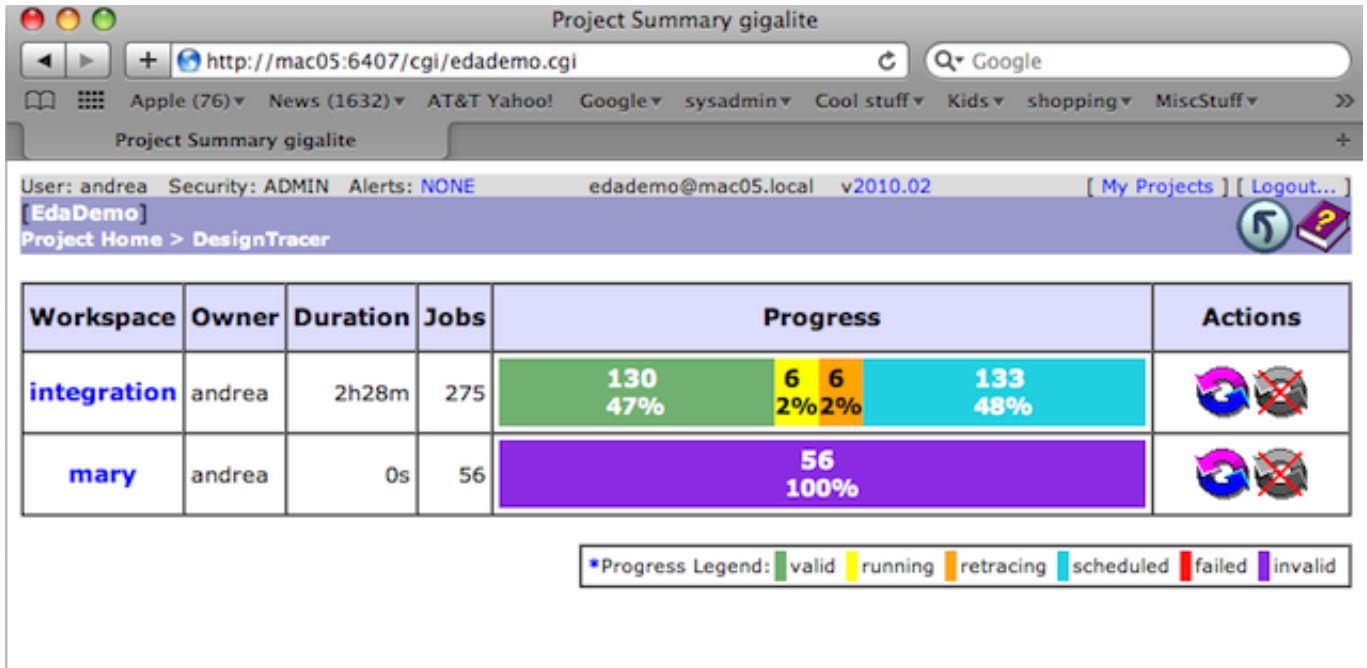


Figure 15: The summary page shows the status of the workspaces

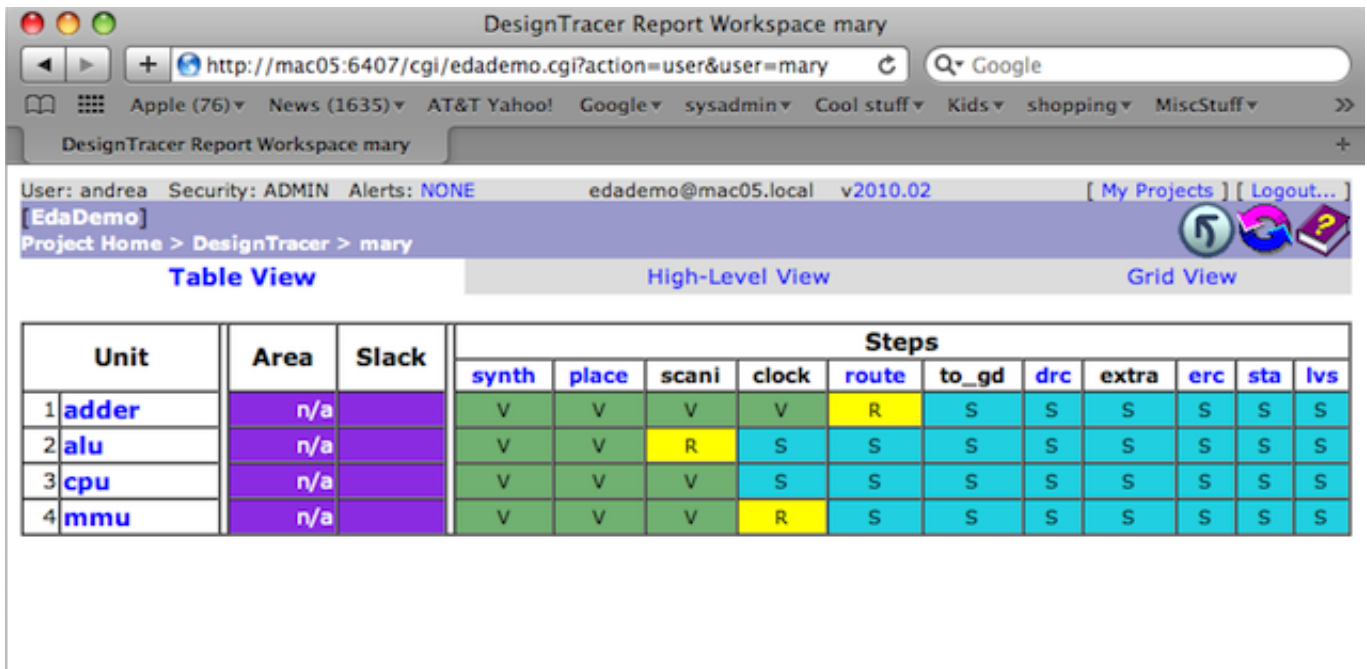


Figure 16: The user page in this EDA-Demo shows a matrix that represents the status of jobs in a specific workspace. There is a row for each block in the design and a column for each of the important jobs in the flow. The color of the cell shows the status of the job.

Here you can see what to expect if you use the VOV Console to run this demo. In particular, you can see the graph view for user 'integration' and the Hi-Level Flows.

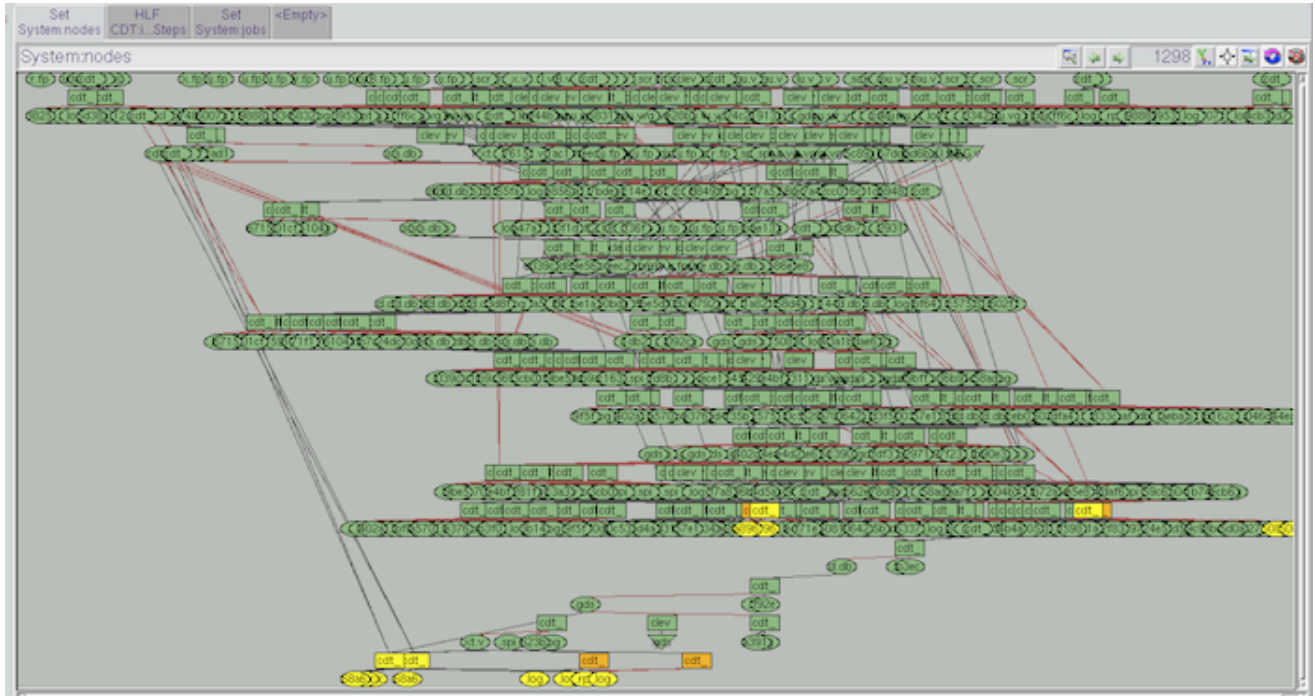


Figure 17: The console shows the complexity of this flow

The Hi-Level Flow representation uses big blocks to represent large sets of jobs. Each block can be clicked on to expose the status of the individual jobs in the set. A little histogram on the bottom left of each block shows the status distribution of the jobs in the set.

2. From this point on, you should be able to:
 - a) Navigate the browser interface
 - b) Run either parts of the flow, or the whole flow
 - c) Stop running jobs
 - d) Monitor the execution of jobs
 - e) Diagnose the failure of jobs
 - f) Edi input files, check them into CVS, and check out any version of a file.



Figure 18:

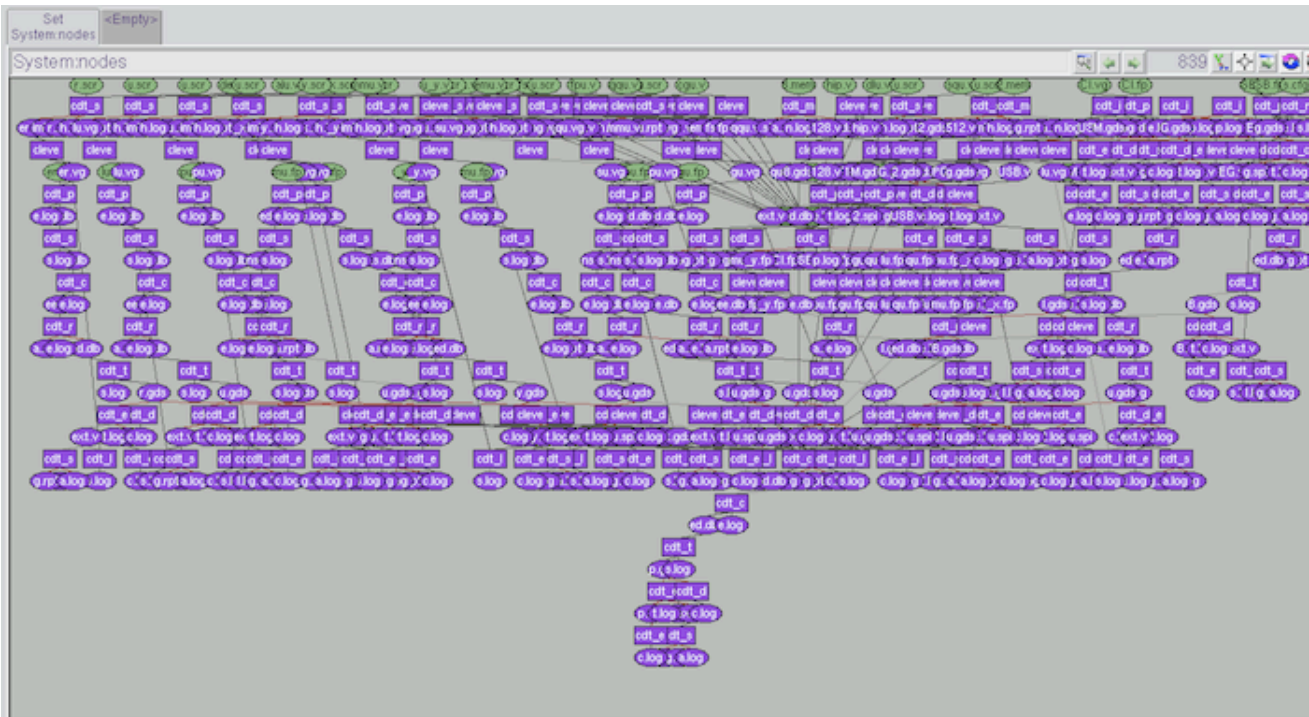


Figure 19:

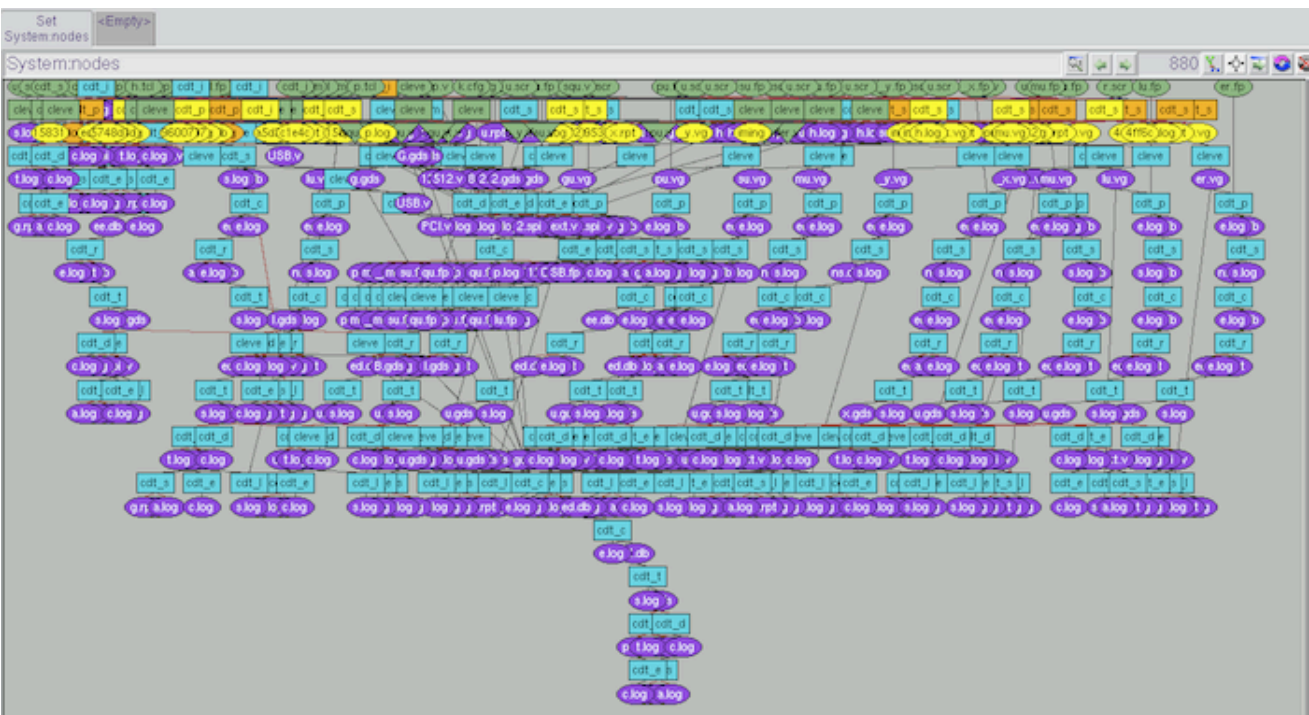


Figure 20:

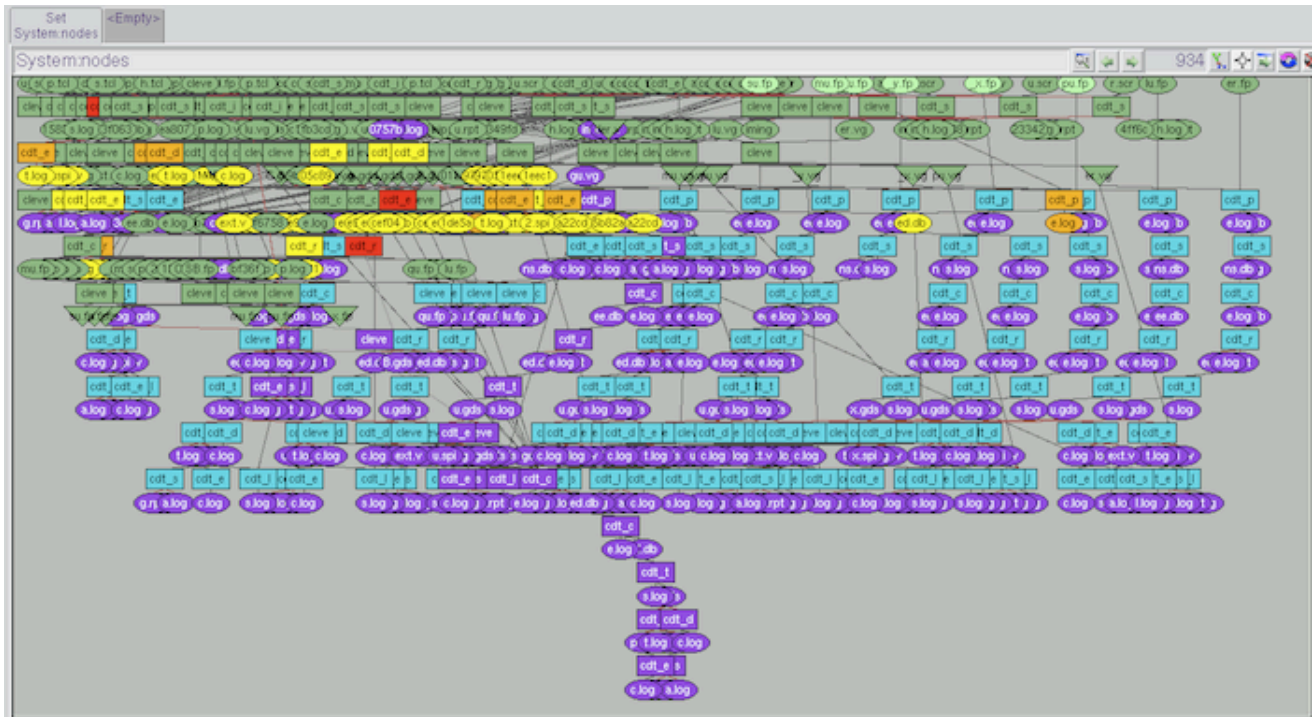


Figure 21:

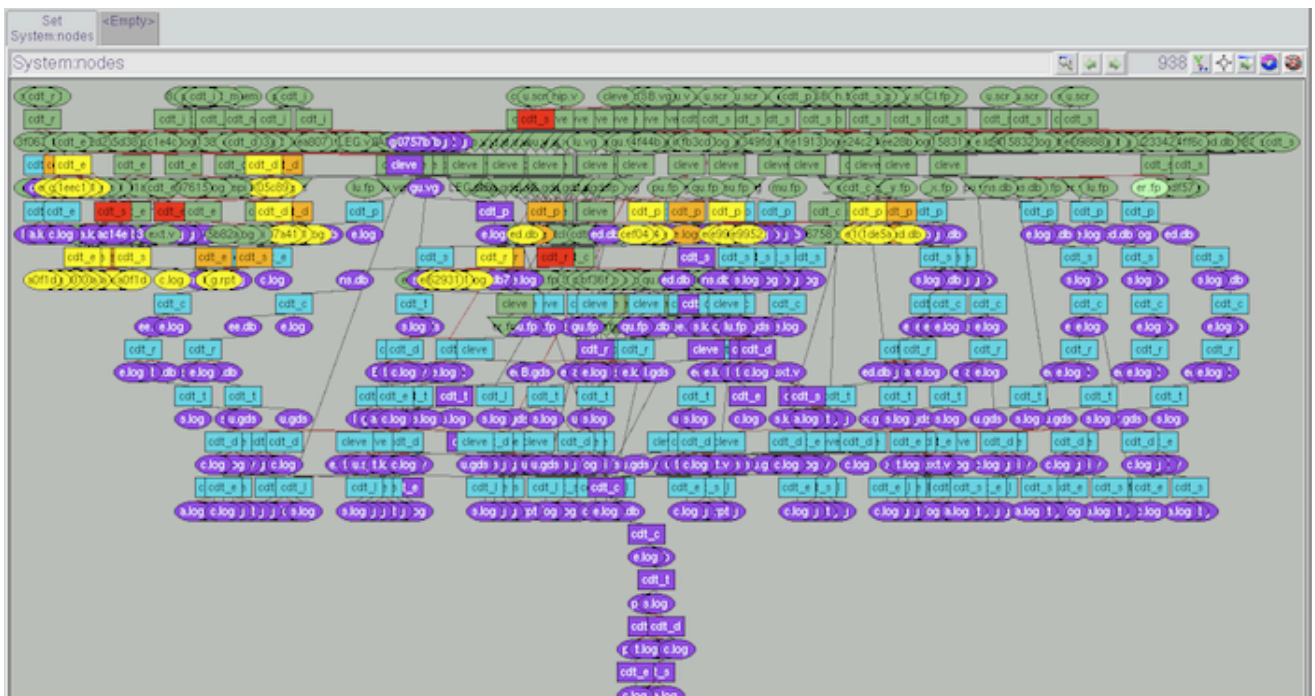


Figure 22:

EDA Demo Part 2: Dissect the Demo

The Chip Structure File and cdt Script

Each design organization has its own way of describing the structure of a chip. For the demo, we have chosen to describe not so much the structure of the chip as much as the type of each block in the chip. We have RTL block, which need to be synthesized, we have memory blocks, and both soft and hard IP cores.

chipstruct.tcl

```
set listOfUnits {
  adder rtl
  alu rtl
  cpu rtl
  mmu rtl
  mmu_x rtl
  mmu_y rtl
  emu rtl
  fsu rtl
  dlu rtl

  mem12x128 memory
  mem32x512 memory
  PCI softip
  USB softip
  ring pads
  chip toplevel
}
```

In a similar way, each design organization has its own set of scripts to invoke the steps in the design flow.

The Capsules

We need a capsule for each tool. Basically, a capsule is just a script that figures out the inputs and outputs of an invocation of the tool. In the following example, we show the capsule for the tool `cdt_synth`. The file name of a capsule should be in the form of `vov_tool_name.tcl`. In this case, the capsule file for `cdt_synth` is named `vov_cdt_synth.tcl`.

The usage of this tool is:

```
% cdt_synth unit
```

It takes two files as inputs:

- `rtl/unit.v`
- `rtl/unit.scr`

and it generates four output files:

- `results/unit.vg`
- `results/unit.timing`

- results/unit.rpt
- results/unit.synth.log

So the capsule is as simple as the following:

vov_cdt_synth.tcl

```
set unit [shift]

VovInput rtl/$unit.v
VovInput rtl/$unit.scr

VovOutput results/$unit.vg
VovOutput results/$unit.timing
VovOutput results/$unit.rpt

VovOutput logs/$unit.synth.log
```

The Flow Description BlockFlow.tcl

This flow creates all jobs for each block (here called unit) described in the chipStruct.tcl file, taking into account the type of the block. The FDL procedure S is used to define sets of jobs, which are later used to generate CGI reports.

BlockFlow.tcl: The Detailed Flow

```
set PROJECT $env(PROJECT)
set USER [file tail [pwd]]

source $env(EDADEMO)/chipStruct.tcl
if [file exists local/chipStruct.tcl] {
    VovMessage "Sourcing local/chipStruct.tcl"
    source local/chipStruct.tcl
}

S "CDT:$USER" {
    foreach { unit type } $listOfUnits {
        E EDADEMO
        lappend allUnits $unit
        set types($unit) $type
        S "CDT:$USER:unit:$unit" {
            indir -create $PROJECT/units/$unit {
                file mkdir netlists

                switch $type {
                    "rtl" {
                        indir -create synthesis {
                            file mkdir results
                            J vw cdt synth $unit
                            J clevercopy results/$unit.vg ../netlists
                            J clevercopy rtl/$unit.v ../../../../data/rtl
                        }
                    }
                }
            }
        }
        switch $type {
            "rtl" - "softip" {
                indir -create place {
                    J vw cdt place $unit
                    J vw cdt scanins $unit
                }
            }
        }
    }
}
```


The CGI script edademo.cgi

This is the most complex piece of the demo. It is also the least necessary, given that we have already covered all pieces necessary to efficiently completing a FlowTracer based design using the CLI and the GUI.

Let's start from the main part of the code, which looks quite similar to the CGI scripts that we have already seen in the CGI tutorial. In bold, you can notice the parsing of QUERY_STRING, which is needed to determine which page to display, based on the values of the "user" and "action" variables.

edademo.cgi: Main code

```
#!/bin/csh -f
# The rest is -*- Tcl -*- \
exec vovsh -t $0 $*

## Definition of all procedures

##### MAIN CODE

VOVHTML_START

set listOfUnits {}
set listOfSteps { synth place route sta drc lvs }
set listOfUsers {}
set results() "Initialization of results array"

set opt(reload) 0
set opt(user) ""
set opt(mode) ""
set opt(action) "summary"

foreach option [split $env(QUERY_STRING) &] {
    if [regexp {(.*)=(.*)} $option all var value] {
        set var [string tolower $var]
        set opt($var) [url_decode $value]
    }
}

switch $opt(action) {
    "job" {
        showJobReport $opt(id)
    }
    "user" {
        getResultsJobs $opt(user)
        getResultsArea $opt(user)
        getResultsTiming $opt(user)
        showUserReport $opt(user) $opt(mode) $opt(reload)
    }
    default {
        showProjectSummary $opt(reload)
    }
}

VOVHTML_FINISH
```

getResultsArea

In the procedure `getResultsArea` we search the trace database for files that contain area information, which in our case are files with a suffix `area.rpt`. We create a temporary set of all such files (with `vtk_set_create`), and then we get all

elements in the set (with `vtk_set_get_elements`). We look inside each of the files (with `source $namex`) and store the area information in the results array, taking care of flagging whether the data up-to-date or not, by looking at the status (`@STATUS@`) of the report file. Finally, we cleanup by forgetting the temporary set.

edademo.cgi: collect data from report files:

```
proc getResultsArea { userToReport } {
  global results
  set setName "@@:tmp:grarp[pid]"
  set setRule "isfile name~/${userToReport}/.*area.rpt$"
  set setId [vtk_set_create $setName $setRule]
  foreach fileInfo [vtk_set_get_elements $setId "@STATUS@ @NAME@" ] {
    set status [shift fileInfo]
    set name [shift fileInfo]
    set namex [vtk_path_expand $name]
    if [file exists $namex] {
      set unit [file root [file root [file tail $namex]]]
      catch {source $namex}
      set results($unit,area,status) $status
    }
  }
  vtk_set_forget $setId
}
```

getResultsJobs

In the procedure `getResultsJobs` we query the trace database for information about the status, duration, etc. about each of the CDT jobs in the flow. We store the information in the results array.

First we get a list of all the sets in the trace (with `vtk_trace_list_sets`). We are interested only in the sets with name beginning with "CDT:". We get the elements of each set (with `vtk_set_find` and `vtk_set_get_elements`).

edademo.cgi: collect data from trace

```
proc getResultsJobs { userToReport } {
  global results
  global listOfUnits
  global listOfUsers

  foreach setName [vtk_trace_list_sets] {
    # puts '$setName'
    if [regexp {^CDT:(.*):unit:([:^:]+)$} $setName all user unit] {
      lappend_no_dup listOfUnits $unit
      lappend_no_dup listOfUsers $user

      if { $user != $userToReport } continue

      set setId [vtk_set_find $setName]
      set results($unit,$setId) $setId
      foreach jobInfo [vtk_set_get_elements $setId "@ID@ @STATUS@ @COMMAND@" ] {
        set id [shift jobInfo]
        set st [shift jobInfo]
        set tool [lindex $jobInfo 1]

        if { $tool == "cdt" } {
          set step [lindex $jobInfo 2]
          set results($unit,$step,$id) $id
          set results($unit,$step,$st) $st
        }
      }
    }
  }
}
```

```
    }
    if [regexp {^CDT:(.*):step:([\^:]+)$} $setName all user step] {
        if { $user != $userToReport } continue
        set setId [vtk_set_find $setName]
        set results(step,$step,setId) $setId
    }
}

set listOfUnits [lsort $listOfUnits]
set listOfUsers [lsort $listOfUsers]
}
```

Page Layout

To give all pages a similar look and functionality, we wrote procedure EDADEMOPAGE, which takes the following arguments:

- *title*, which is the title of the page
- *reloadMs*, the value in milliseconds of the auto-reload period
- *menu_script*, a script to customize the menu on the left hand side of the page
- *body_script*, a script with the actual content of the page

The scripts are evaluated using the Tcl command `uplevel`.

`edademo.cgi`: The page layout

```
proc EDADEMOPAGE { title reloadMs menu_script body_script } {
    global env
    HTML {
        HEAD { omitted }
        BODY {
            TABLE CELLSPACING=0 CELLPADDING=6 BORDER=0 WIDTH="100%" {
                TR BGCOLOR="white" {
                    omitted title code
                }
                TR {
                    TD BGCOLOR="$bgColor" VALIGN=TOP {
                        HREF "/cgi/edademo.cgi" "Summary"; BR
                        if { $menu_script != {} } {
                            uplevel $menu_script
                        }
                    }
                    TD VALIGN=TOP COLSPAN=2 BGCOLOR="#BBCCBB" {
                        uplevel $body_script
                    }
                }
            }
        }
    }
}
```

Following is a collection of procedure used to render the data in HTML. Note the use of `vtk_time_pp` to print durations in a human readable form and of the global array `vov_color()` to colorize the cells in the tables according to the status of the node they represent.

`edademo.cgi`: Data rendering

```
proc showStepInfo { unit step displayMode } {
    global results
```

```

global vov_color

set st [getResult $unit,$step,st EMPTY]
if { $st == "EMPTY" } {
    TD {}
    TD {}
    TD {}
    return
}
set du [getResult $unit,$step,du n/a]
set id [getResult $unit,$step,id 0 ]
set ag [getResult $unit,$step,ag 0 ]

set url  "/cgi/edademo.cgi?action=job&id=$id"
if { $du >= 0 } {
    set tim [vtk_time_pp $du]
} else {
    set tim [vtk_time_pp $ag]
}

if { $displayMode == "simple" } {
    TD ALIGN="right" COLSPAN="2" { SMALL { HREF $url $tim } }
    TD BGCOLOR=$vov_color($st) { OUT " " }
} else {
    TD ALIGN="right" { SMALL { HREF $url $tim } }
    TD ALIGN="center" { SMALL { HREF $url [string tolower $st] } }
    TD BGCOLOR=$vov_color($st) { OUT " " }
}
}

proc showResults { user { displayMode "simple" } } {
    global results
    global vov_color
    global listOfUnits
    global listOfSteps

    TABLE BORDER="0" CELLSPACING="2" CELLPADDING="5" {
        TR BGCOLOR="white" {
            TH COLSPAN=2 { OUT "Unit" }
            TH { OUT "Area" }
            TH { OUT "Slack" }
            foreach step $listOfSteps {
                set setId [getResult step,$step,setId 0]
                TH COLSPAN=3 {
                    if { $setId == 0 } {
                        OUT "$step"
                    } else {
                        HREF "/set?id=$setId&action=showgrid" $step
                        if { $displayMode != "simple" } {
                            BR
                            SMALL {
                                omitted: some links
                            }
                        }
                    }
                }
            }
        }
    }

    set count 0
    foreach unit $listOfUnits {
        set setName "CDT:$user:unit:$unit"
        set setId [vtk_set_find $setName]
    }
}

```

```

        if { $setId != 0 } {
            set area      [getResult $unit,area      "n/a"]
            set areaSt    [getResult $unit,area,status "INVALID"]
            set slack     [getResult $unit,slack  ""]
            set slackSt   [getResult $unit,slack,status "INVALID"]
            TR {
                TD ALIGN="RIGHT" { OUT [incr count] }
                TH ALIGN="LEFT" { HREF "/set?id=$setId" $unit }
                TD ALIGN="RIGHT" BGCOLOR=$vov_color($areaSt) {
                    COLOR $vov_color($areaSt,fg) $area
                }
                TH ALIGN="RIGHT" BGCOLOR=$vov_color($slackSt) {
                    if { $slack != "" } {
                        if { $slack %lt; 0 } {
                            COLOR "#FFFF88" "($slack)"
                        } else {
                            COLOR $vov_color($slackSt,fg) $slack
                        }
                    }
                }
            }
            foreach step $listOfSteps {
                showStepInfo $unit $step $displayMode
            }
        }
    }
}

proc showUserSummary { user setId } {
    global vov_color

    if { $setId == 0 } {
        set status    EMPTY
        set nodes     0
        set places    0
    } else {
        set setInfo [vtk_set_statistics $setId]
        set saveSetInfo $setInfo
        set status   [shift setInfo]
        set nodes    [shift setInfo]
        set places   [shift setInfo]
        set jobs     [shift setInfo]
        set duration [shift setInfo]
        set unknown  [shift setInfo]
        set placeStats [shift setInfo]
        set jobStats [shift setInfo]

        set statusList { INVALID RUNNING VALID FAILED }
        foreach s $statusList { set js($s) 0 }
        foreach { s n } $jobStats {
            set js($s) $n
        }

        TR {
            TH { print user name }
            TD ALIGN="RIGHT" { OUT [vtk_time_pp $duration] }
            TD ALIGN="RIGHT" { OUT $jobs }
            foreach s $statusList {
                omitted: show colored ball
            }
        }
    }
}

```

```
}
```

showProjectSummary

The `showProjectSummary` procedure displays the summary page. First we get the list of users of the project (really the list of workspaces) by looking at the sets with name beginning with "CDT:", and then we call `showUserSummary` on each.

edademo.cgi: The project summary page

```
proc showProjectSummary { reloadMs } {
  set listOfUsers {}

  foreach setName [vtk_trace_list_sets] {
    if [regexp {CDT:(.*):step} $setName all user] {
      lappend_no_dup listOfUsers $user
    }
  }
  set listOfUsers [lsort $listOfUsers]

  EDADEMOPAGE "Project Summary $env(PROJECT)" $reloadMs {
    omitted
  } {
    TABLE WIDTH="100%" border=0 CELLPADDING=10 {
      TR BGCOLOR="#DDDDFF" {
        foreach head {
          Workspace Duration Jobs Invalid
          Running Valid Failed Actions
        } {
          TH { OUT $head }
        }
        foreach user $listOfUsers {
          showUserSummary $user [vtk_set_find "CDT:$user"]
        }
      }
      TR BGCOLOR="#DDDDFF" {
        TD COLSPAN=9 { OUT "Totals" }
      }

      showUserSummary "" [vtk_set_find "System:nodes"]
    }
  }
}
```

showUserReport

The procedure `showUserReport` generates the page showing the jobs in a user workspace. Notice the use of `EDADEMOPAGE` and `showResults`, which have been explained above.

edademo.cgi: The user workspace report

```
proc showUserReport { user mode reloadMs } {
  global argv

  EDADEMOPAGE "EDA-Demo Report Workspace $user" $reloadMs {
    if { $mode != "hlf" && $mode != "llf" } {
      omitted: autoreload control
    }
    showMenu $user $mode
  } {
    switch $mode {
```

```
        "llf" { showLowLevelFlow "$user" }
        default { showResults $use }
    }
}
```

This chapter covers the following:

- [Create a FlowTracer Project](#) (p. 121)
- [Flow Description Language](#) (p. 151)
- [Stop the Project](#) (p. 161)

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 &
```

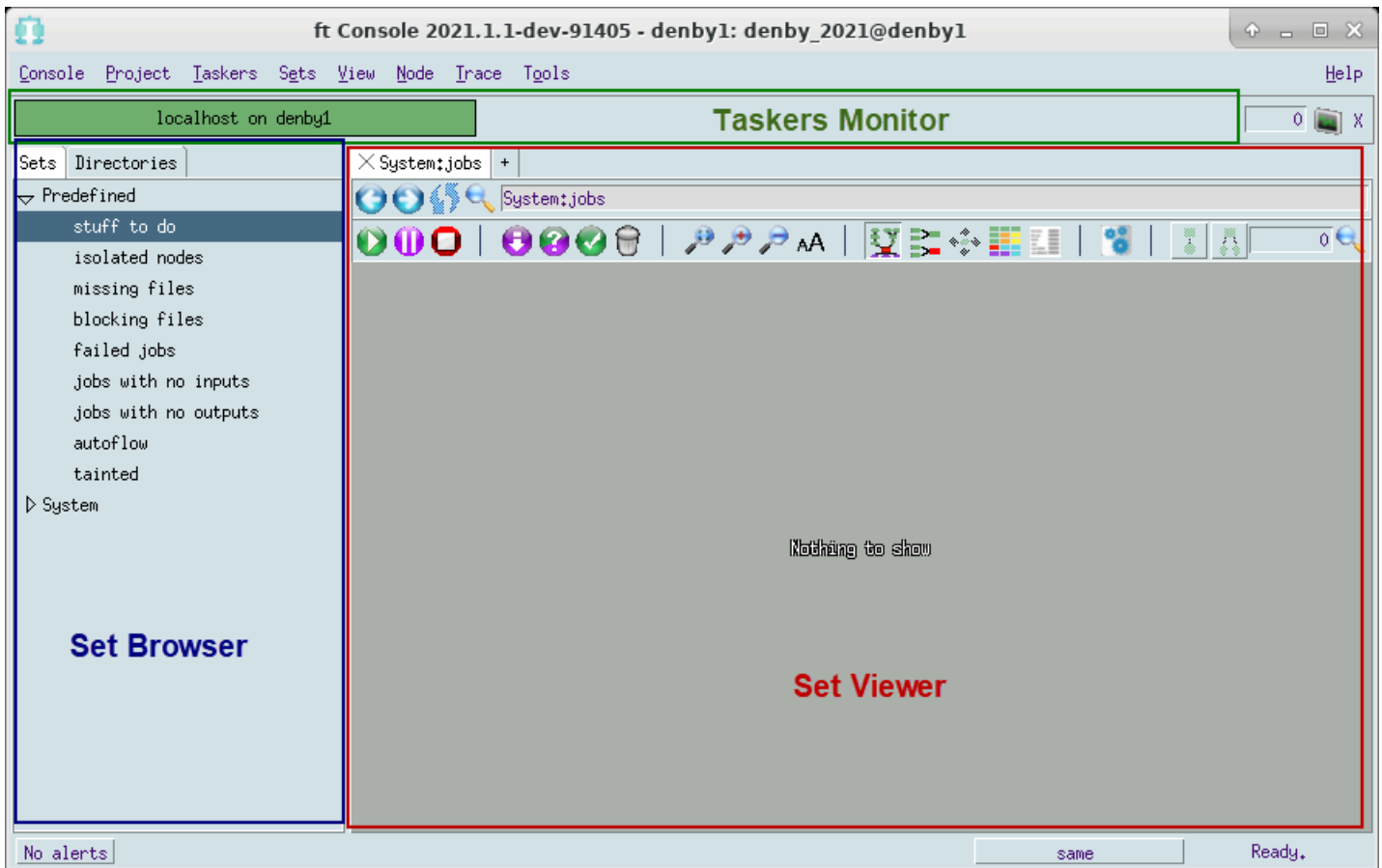


Figure 23: Initial console screen

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. For further information, please see [Excluding Files from the Graph](#).

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.

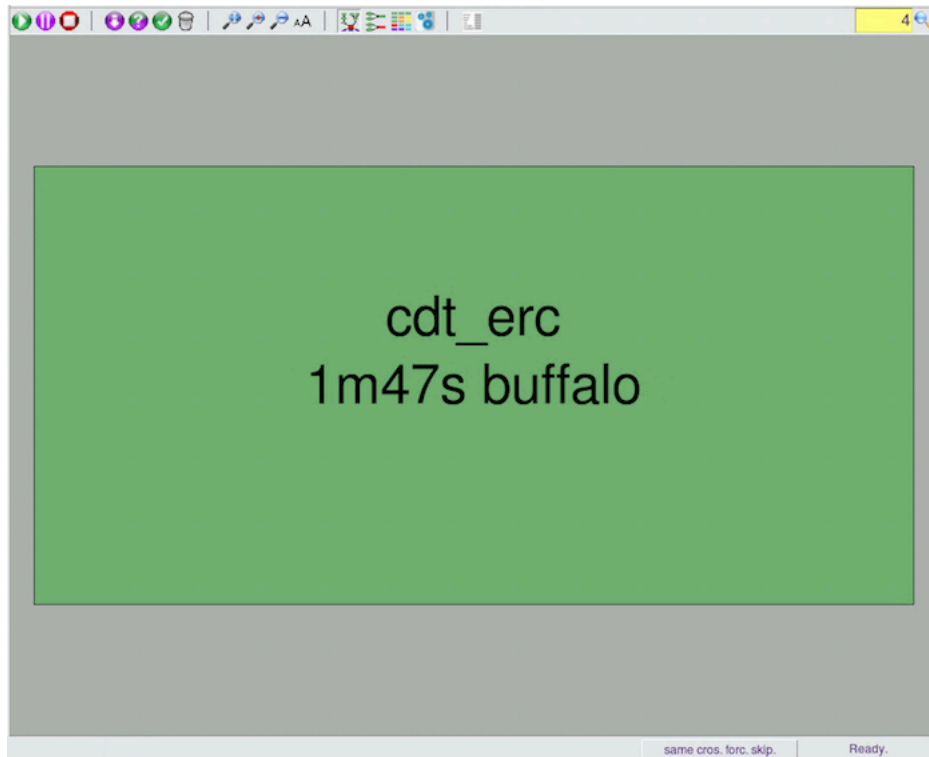


Figure 24:

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.

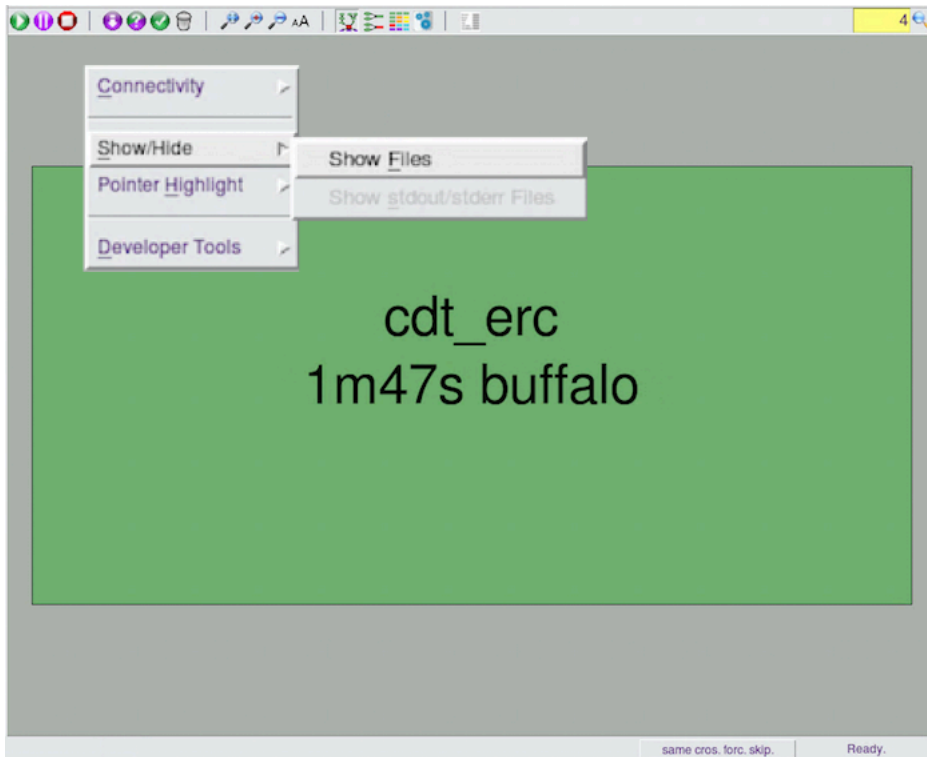


Figure 25:

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.

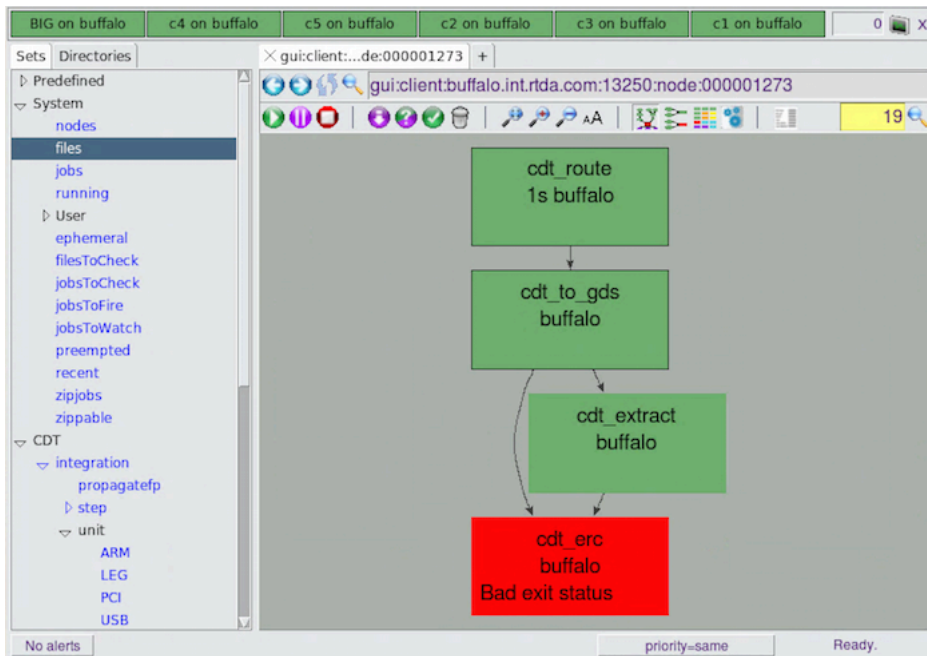




Figure 26:

 **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.

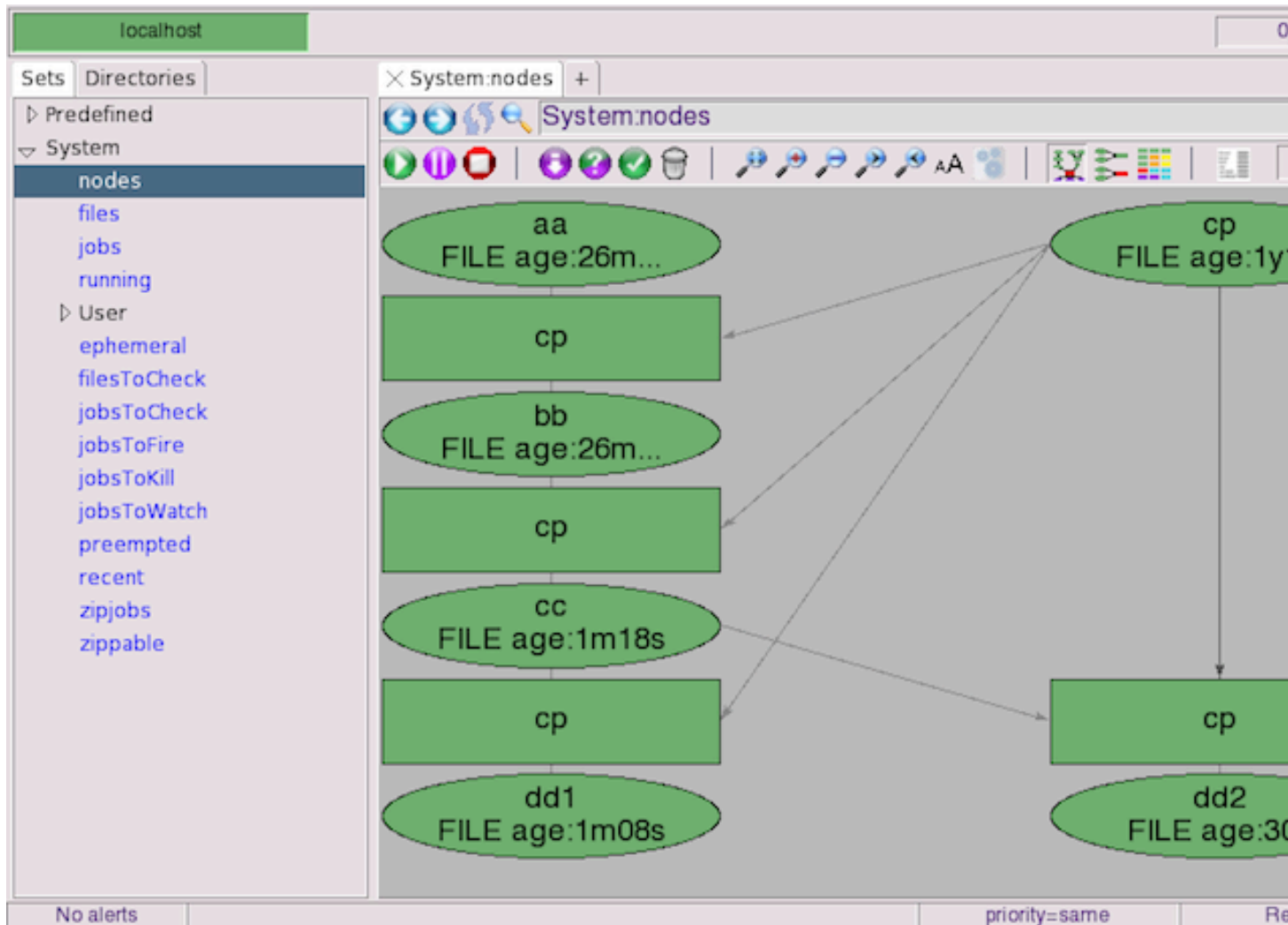


Figure 27:

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.

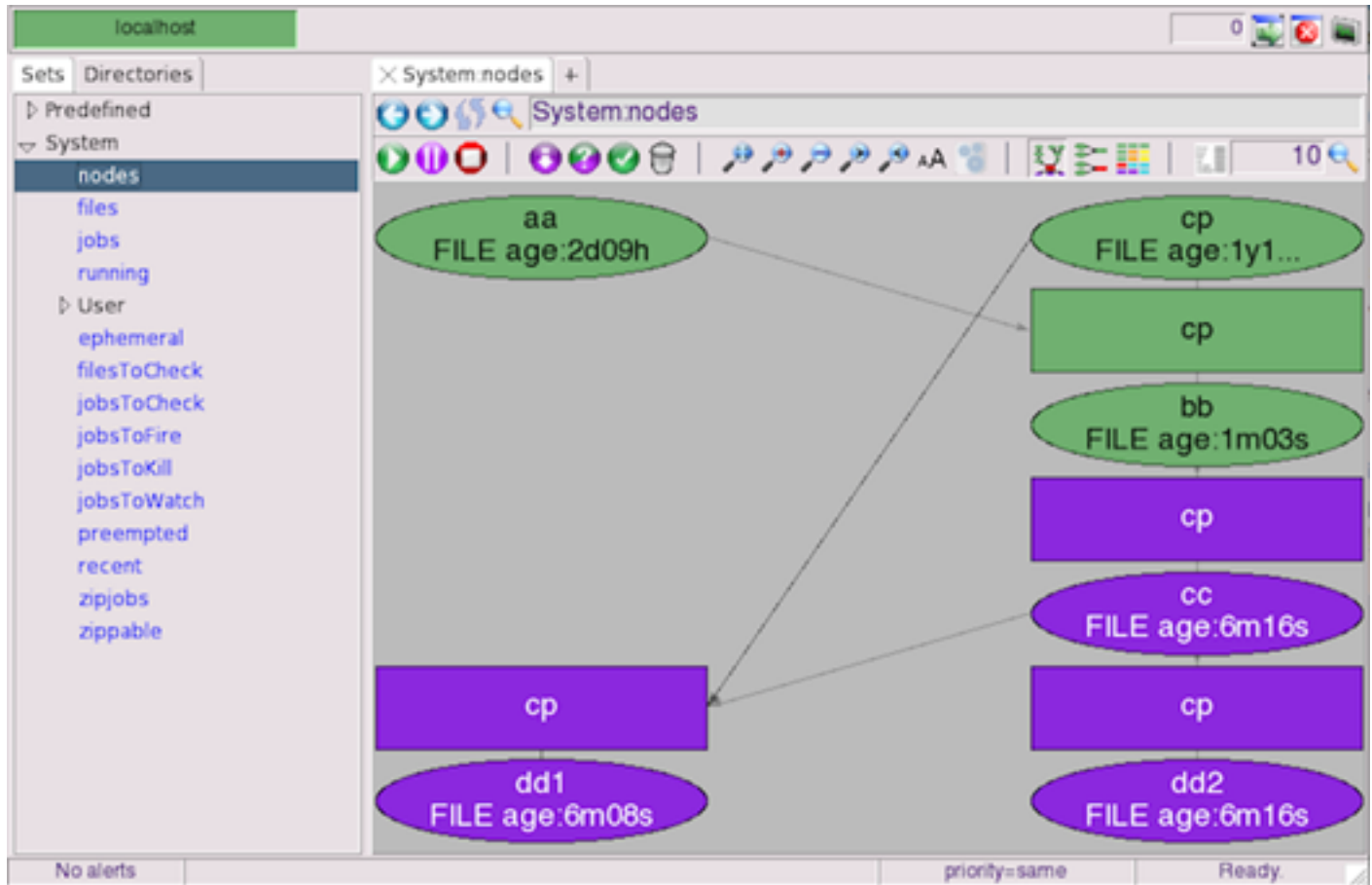



Figure 28:

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.

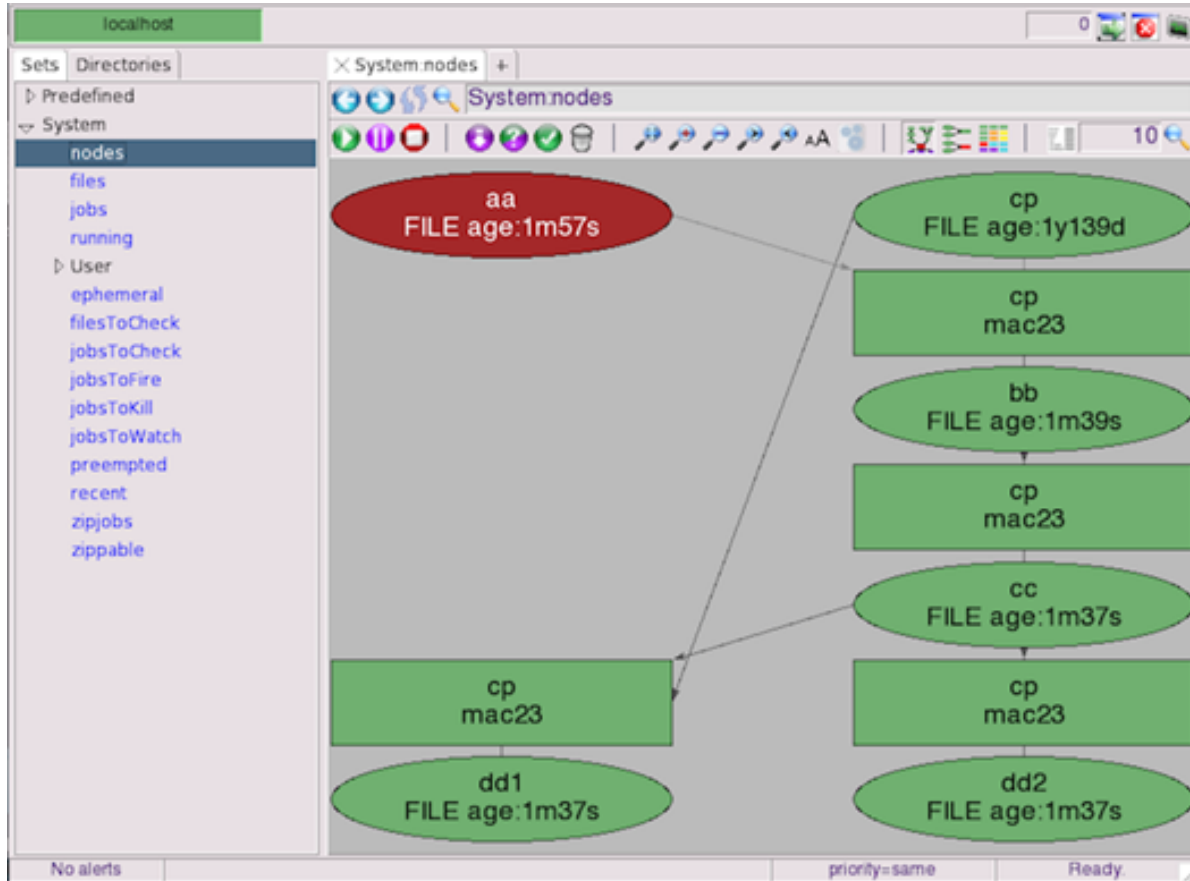


Figure 29:

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.

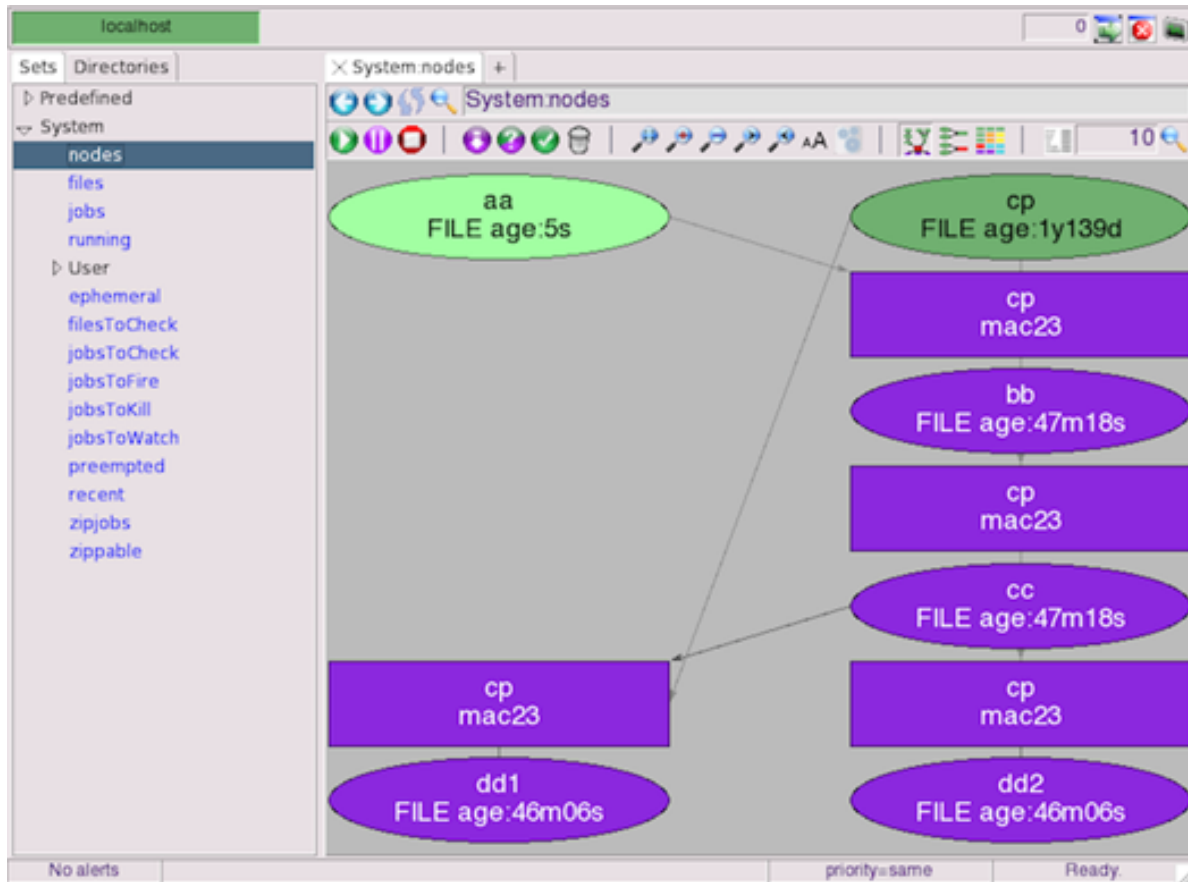


Figure 30:

3. Click **Refresh** > **All Tabs** to update the display and to change the node color of "aa" back to the normal green. You can now see that the VALID nodes have the standard green color and the dependent nodes that are INVALID are purple.

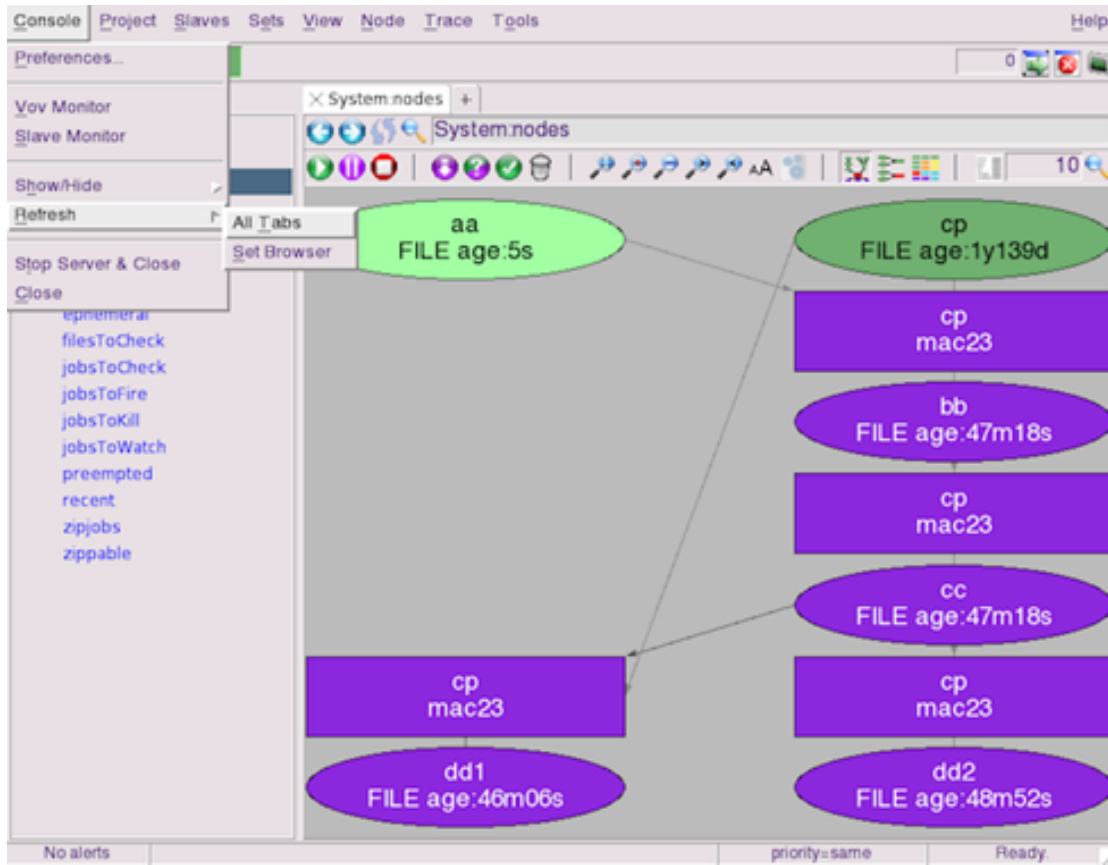


Figure 31:

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

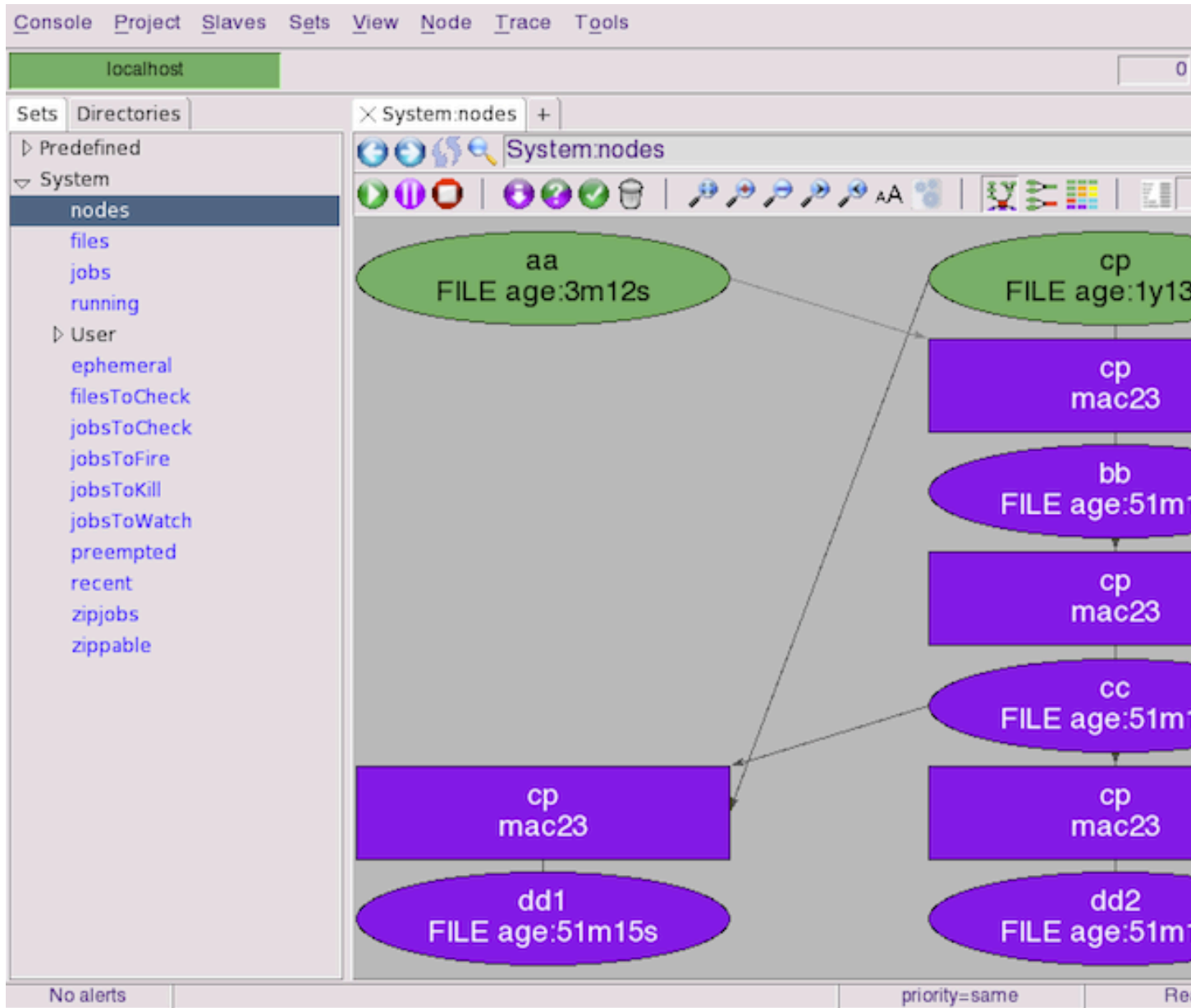


Figure 32:

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

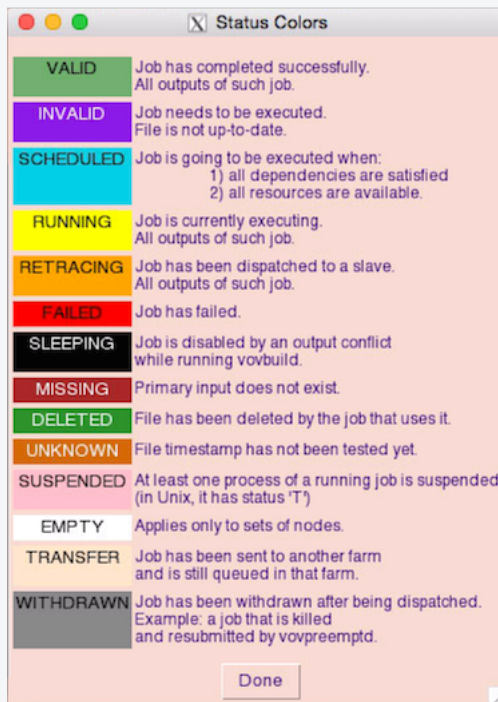


Figure 33:

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.

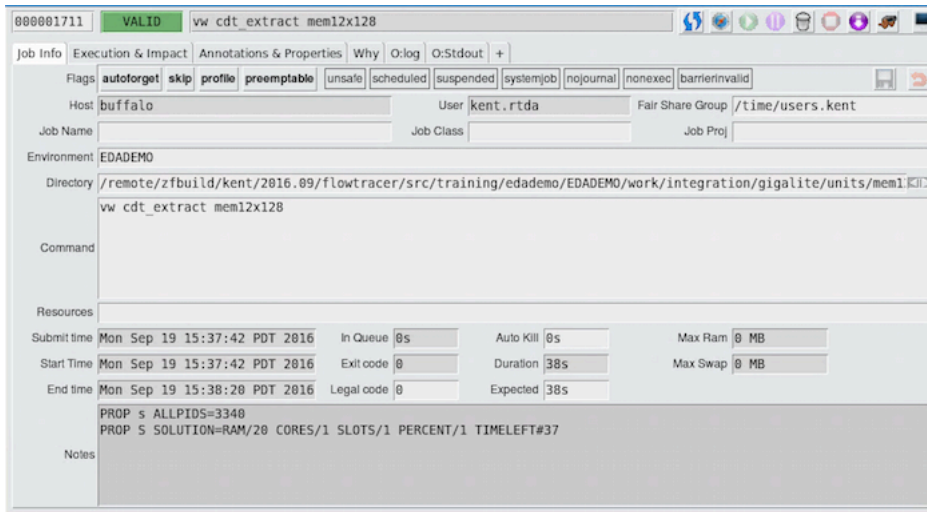


Figure 34:

- 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**.

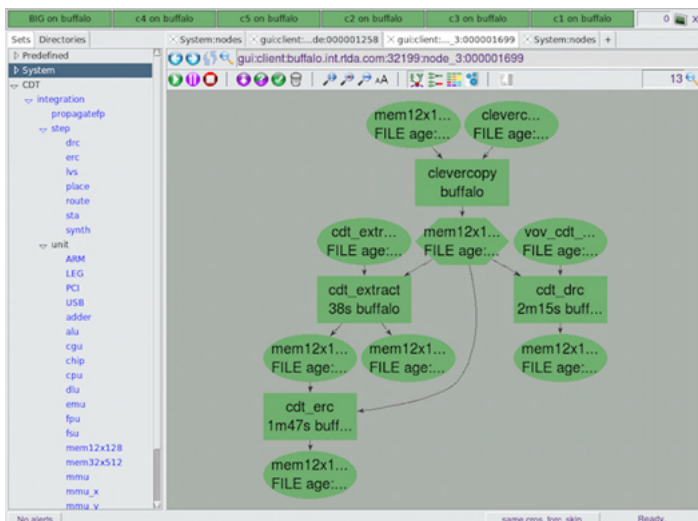


Figure 35:

Horizontal Graph View

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

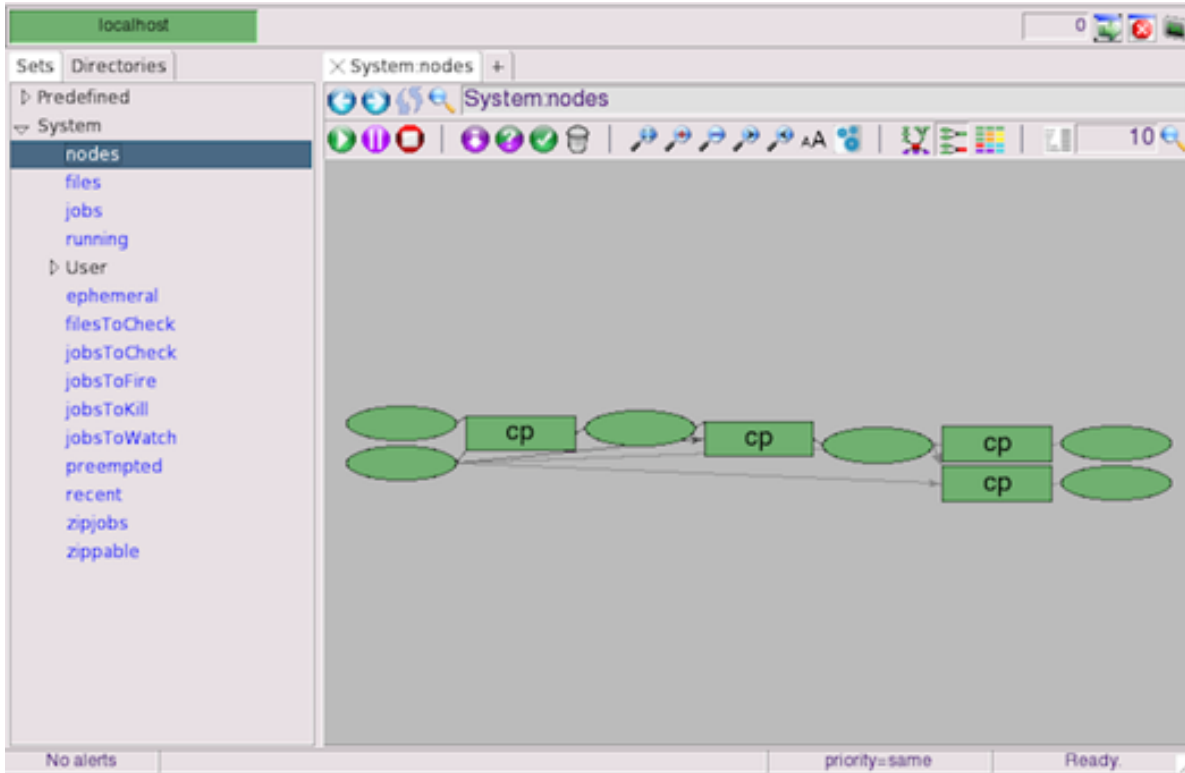


Figure 36:

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 37:

Stat View

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

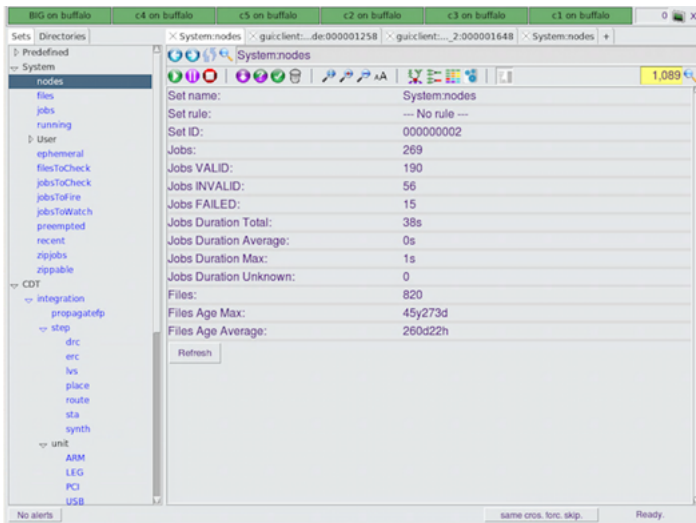



Figure 38:

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:

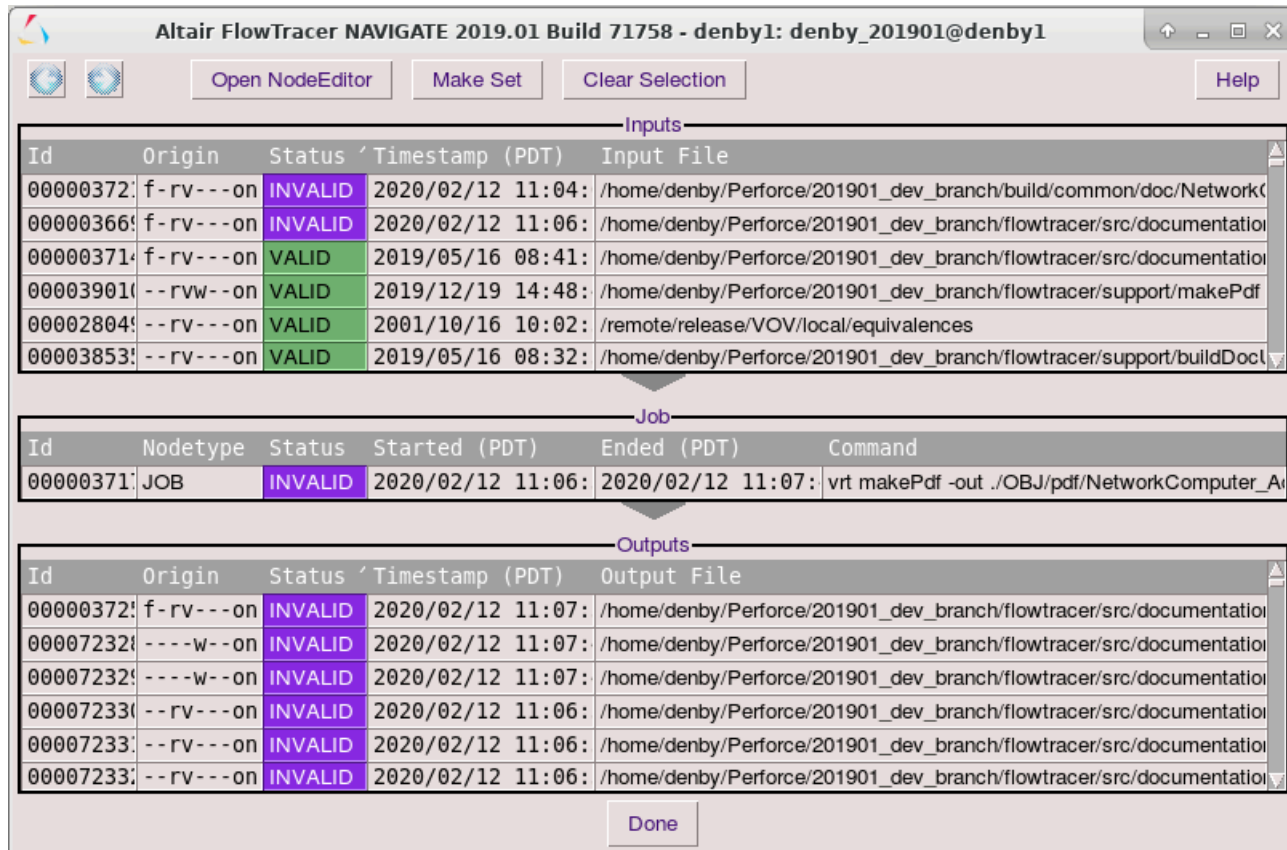



Figure 39:

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:

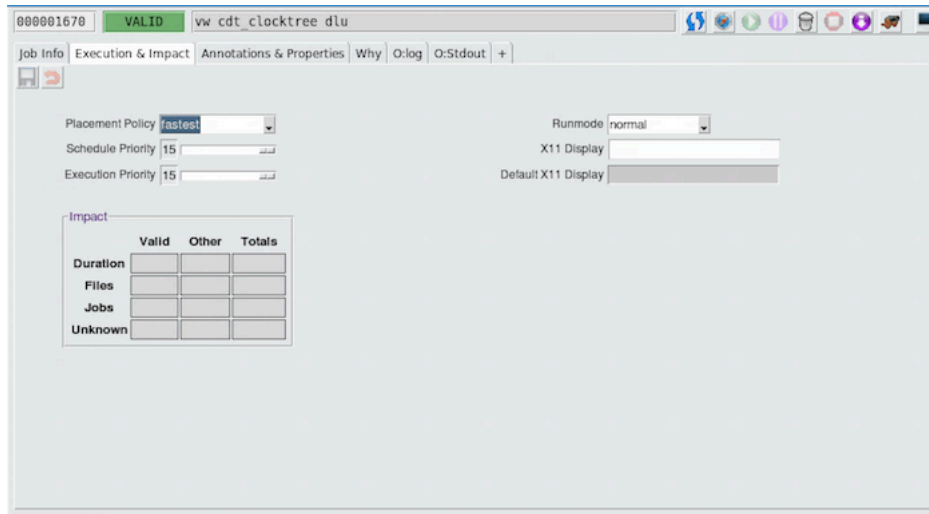


Figure 40:

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.

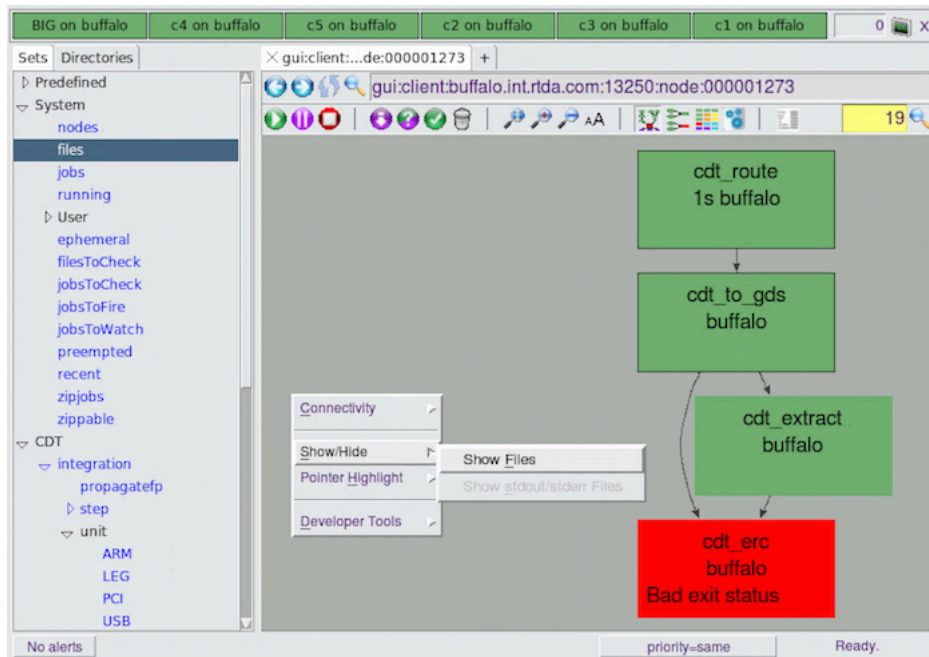


Figure 41: Context menu seen when right-clicking in open area of canvas in Set Viewer

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.

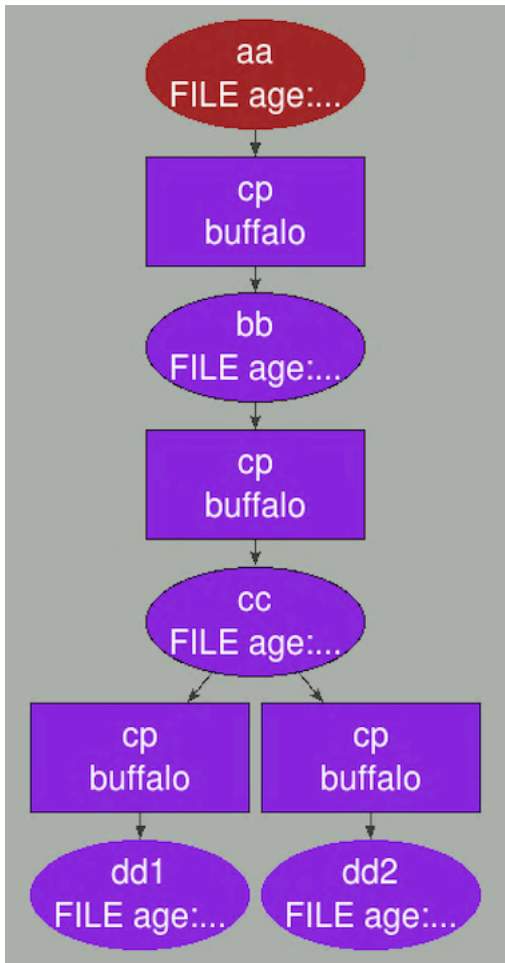


Figure 42:

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.

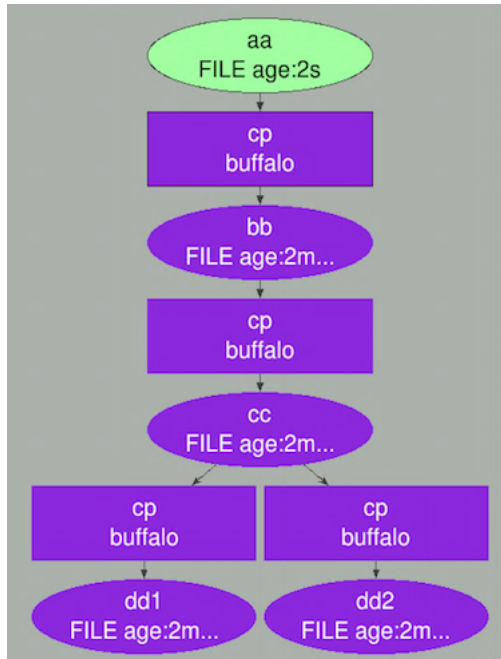


Figure 43:

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
```

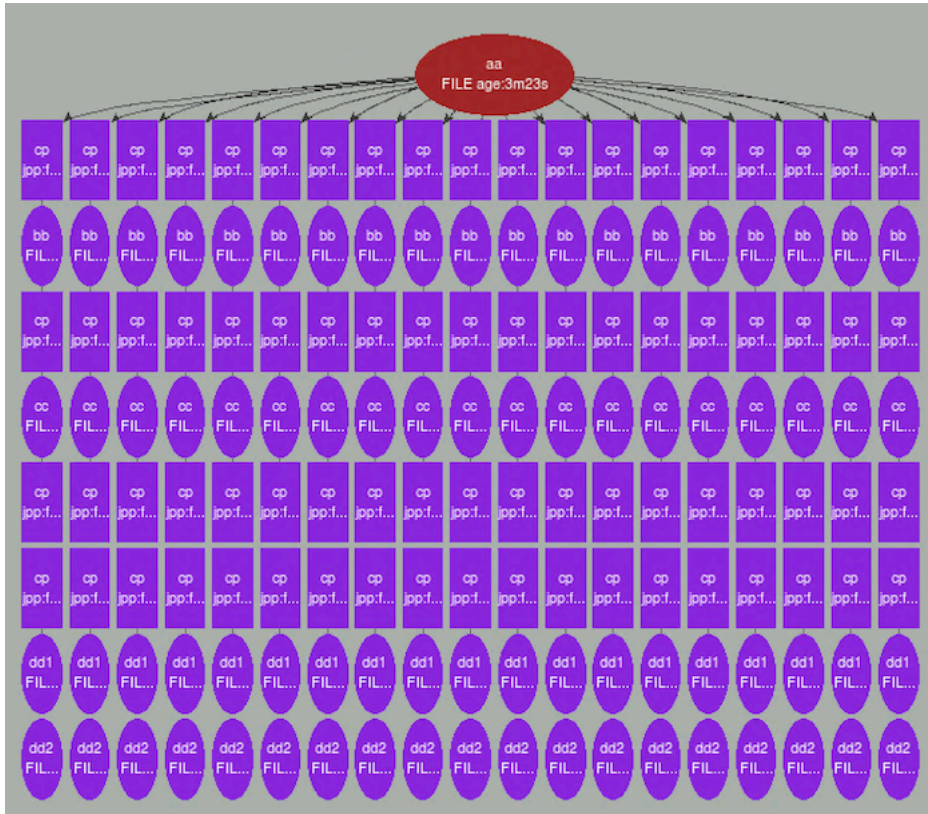


Figure 44:

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.

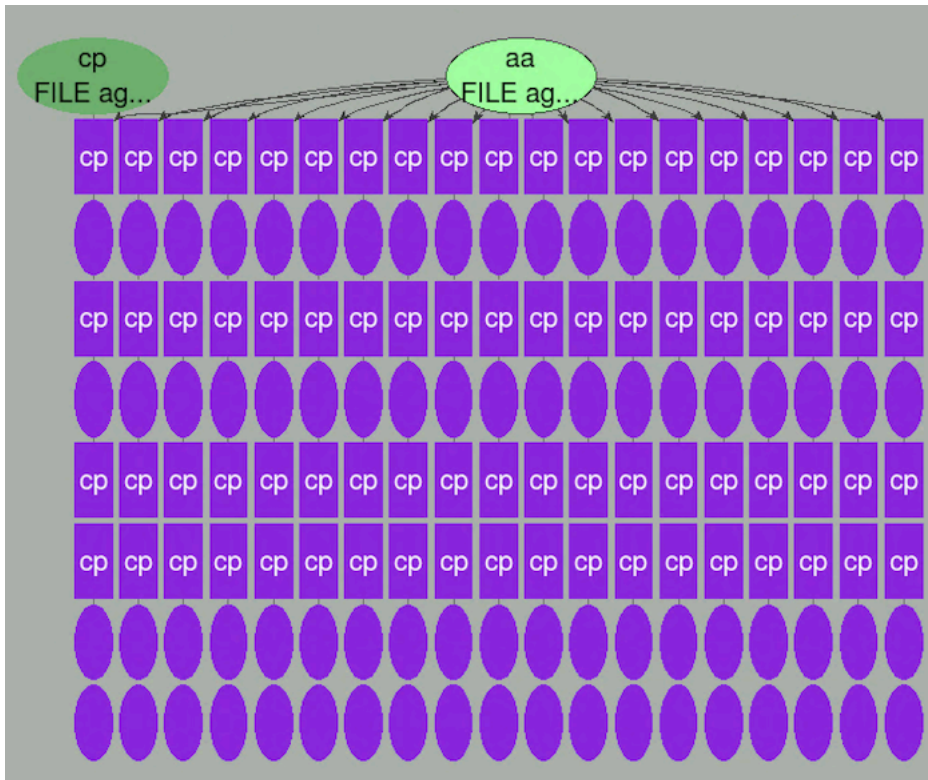


Figure 45:

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. 163)
- [Write Flows](#) (p. 165)

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
```

Generate Custom HTML Reports Using CGI Tutorial

This chapter covers the following:

- [Job Reports: List](#) (p. 170)
- [Job Reports: Table](#) (p. 171)

Every vovserver has a built-in HTTP interface, which includes the ability to quickly generate customized reports using the CGI (Common-Gateway-Interface) mechanism. This tutorial guides you through the first simple CGI script.

You will write a CGI script that is project specific, and install it in the cgi subdirectory of the server configuration directory:

```
% set cgidir = `vovserverdir -p cgi`
% echo $cgidir
/home/someuser/vov/test.swd/cgi # Or similar
% mkdir $cgidir
% cd $cgidir
% vi tutorial.cgi
```

File tutorial.cgi

```
#!/bin/csh -f
# The rest is -*- Tcl -*- exec vovsh -f $0 $*

# This is file tutorial.cgi

VOVHTML_START
HTML {
    HEAD { TITLE "CGI Tutorial" }
    BODY {
        OUT "Hello World"
    }
}
VOVHTML_FINISH
```

The first 3 lines are a common UNIX technique to make this script executable by vovsh, which is the main FlowTracer client. Notice that this script is written in Tcl syntax.

The commands VOVHTML_START and VOVHTML_FINISH are required.

The procedures HTML, HEAD, BODY, and OUT, are defined in the file \$VOVDIR/tcl/vtcl/vovhtmlgen.html and are a convenient way to generate well formed HTML code. You are not required to write the CGI script in Tcl, only we believe you will find it extremely more convenient to do, especially considering that the FlowTracer API is itself in Tcl.

To satisfy the curious, here is how the same script could have been written without Tcl.

```
#!/bin/csh -f

# This is the hard way to write a CGI script.
echo "\r" # Important empty line!
echo "<html>"
```



```
echo "<head><title>CGI Tutorial</title></head>"  
echo "<body>Hello World</body>"  
echo "</html>"  
exit 0
```

Make the script executable and use a browser to visit the URL given to you by the command `vovbrowser`.

```
% chmod a+x tutorial.cgi  
% vovbrowser -url /cgi/tutorial.cgi  
http://host:port/cgi/tutorial.cgi
```

Job Reports: List

Use an ordered list

```
#!/bin/csh -f
# The rest is -- Tcl -- exec vovsh -f $0 $*

# This is file tutorial.cgi

VOVHTML_START
HTML {
  HEAD { TITLE "CGI Tutorial" }
  BODY {
    set setId [vtk_set_find "System:jobs"]
    OL {
      foreach job [vtk_set_get_elements $setId "@ID@ @STATUS@ @HOST@" ] {
        LI { OUT $job }
      }
    }
  }
}
VOVHTML_FINISH
```

In this example we introduce two vtk procedures:

1. `vtk_set_find` takes a set name and returns its the set id. In this case, the name of the set is "System:jobs" which is a system set.
2. `vtk_set_get_elements` which takes 2 arguments, namely the id of a set and a format string. In this case, the format string asks for the job id, its status, and the host on which the job was executed.

We also show two more HTML procedures, that is OL and LI, which are used to build ordered lists.

Job Reports: Table

Use a table.

```
#!/bin/csh -f
# The rest is -- Tcl -- exec vovsh -f $0 $*

# This is file tutorial.cgi

VOVHTML_START
HTML {
  HEAD { TITLE "CGI Tutorial" }
  BODY {
    set setId [vtk_set_find "System:jobs"]
    TABLE border="1" align="center" {
      foreach job [vtk_set_get_elements $setId "@ID@ @STATUS@ @HOST@" ] {
        set id [shift job]
        set status [shift job]
        set host [shift job]
        TR {
          TH { OUT $host }
          TD { HREF "/node/$id" $id }
          TD { OUT $status }
        }
      }
    }
  }
}
VOVHTML_FINISH
```

In this example we introduce the HTML procedures TABLE, TR, TH and TD, which are used to create tables. The procedure HREF is used to introduce a hyperlink. In this case, the page "/node/XXX" has a detailed description of the node with id XXX.

Finally, in this example we introduce the Tcl procedure shift which is really an Altair Accelerator extension and is used to take the first element from a list while shifting the list left by one.

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 [herehere](#).

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

Special Characters

.cshrc [20](#)
.profile [20](#)

A

Accelerator Administrator Tutorials [45](#)
access to help [18](#)
add a job interactively [125](#)
add additional environment directories [67](#)
add more jobs to the flow [130](#)
add workstation/offhours vovtaskers [61](#)
advanced commands [72](#)
advanced policy configuration [54](#)
advanced REST usage [93](#)
Allocator Case Study [98](#)
analyze impact [144](#)
Art of Flows Example 1 [6](#)
Art of Flows Example 1 with Scripts [6](#)
Art of Flows Example 2 [9](#)
Art of Flows Example 2 with scripts [9](#)
Art of Flows Example Guide [5](#)

B

batch process to define and run a flow [156](#)
bin/bash [20](#)
bin/csh [20](#)
bin/tcsh [20](#)
browser-based setup [51](#)
build a more complex flow [166](#)
build the flow [153](#), [165](#)

C

change dependent input file [132](#)
check file status [146](#)
check job status [147](#)
check project information [123](#)
command line interface [146](#)
compose good environment scripts [68](#)
configuration examples [56](#), [68](#)
configure an environment [65](#)
configure and manage Monitor-basic [74](#)
configure default resources [54](#)
configure FairShare [52](#)

- configure policy.tcl for FairShare [52](#)
- configure resources [56](#)
- configure security [58](#)
- configure taskers [60](#)
- connection keep-alive [95](#)
- create a complex flow [157](#)
- create a FlowTracer project [121](#)
- create a project [122](#)
- create a project directory [126](#)
- create efficient VOV scripts [163](#)
- customize actions needed to enable access to Altair Accelerator products on Windows [24](#)

D

- define a logical HOME directory name [70](#)
- detect conflicts [148](#)
- determine reason for invalid node status [144](#)

E

- EDA Automation Tutorial [101](#)
- EDA Demo Part 1 [102](#)
- EDA Demo Part 1 - check out the data [103](#)
- EDA Demo Part 1 - customize the project [103](#)
- EDA Demo Part 1 - setup [102](#)
- EDA Demo Part 1 - start the browser interface [104](#)
- EDA Demo Part 1 - start the project [102](#)
- EDA Demo Part 2 [110](#)
- EDA Demo Part 2 - blockflow.tcl [111](#)
- EDA Demo Part 2 - chip structure file and cdt script [110](#)
- EDA Demo Part 2 - edademo.cgi [113](#)
- EDA Demo Part 2 - the capsules [110](#)
- EDA flows [159](#), [166](#)
- enable a shell [123](#)
- enable Altair Accelerator for non-interactive shells [21](#)
- enable CLI access on UNIX [20](#)
- enable CLI access on Windows [23](#)
- enable the command prompt to communicate with a running product server [25](#)
- enable the shell to communicate with a running product server [22](#)
- enforce job resource rules [54](#)
- example application: REST 101 [83](#)
- example applications: job submit and list [83](#)
- execute the flow [166](#)

F

- find and view example environment scripts [65](#)
- find and view resources.tcl file [56](#)

find the server working directory [46](#)
find the URL [51](#)
flow description language [151](#)
flow.tcl file [152](#), [165](#)
FlowTracer Advanced Tutorials [162](#)
FlowTracer Beginner's Tutorial [120](#)
forget nodes and sets from the graph [145](#)

G

generate custom HTML reports using CGI tutorials [168](#)
get detailed information about a job [37](#)
get ready for REST [82](#)
get summary information [28](#)
grid view [140](#)
GUI job views [138](#)

H

help, Accelerator [18](#)
horizontal graph view [140](#)
HTTPS security considerations [95](#)

I

invoke the GUI [41](#)
issuing REST requests from the Swagger web UI [91](#)

J

job control [34](#)
job reports: list [170](#)
job reports: table [171](#)
jobs, show current information [41](#)
JWT access tokens [95](#)
JWT token allocation [96](#)

K

key REST concepts [80](#)

L

launch jobs with non-default options [87](#)
locate server configuration directory to find policy.tcl [52](#)
locate the security configuration file: security.tcl [58](#)
logical names (equivalences [70](#)

M

make changes to a file and run it [142](#)
method 1: use windows explorer to set command line environment [23](#)
method 2: using Windows command prompt to set command line environment [23](#)
metrics, scheduler [41](#)
Monitor Basic setup [73](#)
monitoring jobs, taskers and resources [39](#)

N

navigate the graph [142, 143](#)
nc_gui [41](#)
nc_info.py [94](#)
nc_list.py [83](#)
nc_run.py [83](#)

O

online help [18](#)
overlapping queues [76](#)

P

PDF, access [18](#)

R

register one job from the command line [126](#)
remove depended input file [134](#)
remove older sets [151](#)
repeat the tutorial without the GUI [149](#)
rerun from the CLI [147](#)
rerun jobs [36](#)
resource management [73](#)
resource monitor [57](#)
resource path [80](#)
resource throttling [75](#)
REST request detailed documentation [88, 90](#)
restart the Accelerator queue [50](#)
restore the shell prompt [123](#)
run a flow from the command line [155](#)
run basic jobs [26](#)
run jobs with various options [31](#)
run the flow interactively [154](#)
run the jobs [142](#)

S

save the new configuration [52](#)

- [scheduler metrics](#) [41](#)
- [security configuration examples](#) [58](#)
- [server configuration parameters](#) [53](#)
- [set command line environment](#) [121](#)
- [setup using the web page](#) [51](#)
- [specify name of logfile: -l <logfile>](#) [33](#)
- [start a queue](#) [46](#)
- [start a test queue](#) [46](#)
- [start a vovtasker from the command line](#) [62](#)
- [start Accelerator](#) [49](#)
- [start newly defined vovtaskers](#) [61](#)
- [start the GUI console](#) [124](#)
- [start/stop Accelerator](#) [49](#)
- [start/stop vovtaskers](#) [62](#)
- [stat view](#) [141](#)
- [stop Accelerator](#) [50](#)
- [stop the project](#) [161](#)
- [submit multiple jobs at once](#) [31](#)

T

- [test the new FairShare configuration](#) [53](#)
- [troubleshooting](#) [44](#)
- [troubleshooting the UNIX setup](#) [22](#)

U

- [upgrade Accelerator](#) [76](#)
- [use a specific queue](#) [48](#)
- [use Accelerator help](#) [18](#)
- [use Altair Accelerator's REST API to submit and list jobs](#) [79](#)
- [use environments: -e <env>](#) [31](#)
- [use of resources: -r <res1 res2 ...>](#) [32](#)
- [use the set browser](#) [125](#)
- [use the web browser](#) [43](#)

V

- [verify access to Altair Accelerator products](#) [20](#)
- [verify context is working](#) [24](#)
- [vertical graph view](#) [139](#)
- [view Accelerator status](#) [49](#)
- [view graph subsets](#) [141](#)
- [vov_rest_v3.py Python library module](#) [88](#)
- [VOV_STDOUT_SPEC](#) [18](#)
- [vovbrowser](#) [18](#)
- [vovbuild](#) [18](#)
- [vovconsole](#) [41](#)

vovdoc [18](#), [18](#)

void [18](#)

VOVRestV3 Python class description [88](#)

vovserver [18](#)

vovtasker configuration [60](#)

vtk_flexlm_monitor procedures [75](#)

W

wait for jobs [32](#)

write flows [165](#)