ALTAIR

ONLY FORWARD

Altair FlowTracer 2025.1.2

Administrator Guide

altair.com

Contents

Altair FlowTracer Administrator Guide

This manual is aimed at the person who will be the administrator for FlowTracer. This person will be responsible for configuring the network resources, setting up the licensing, and establishing security and access rules for the system.

This chapter covers the following:

The FlowTracer administrator is expected to have read the *FlowTracer User Guide* and the *FlowTracer Developer Guide* and be quite familiar with the product. The administrator is expected to comfortable with the topics covered in those manuals and understand that they are not repeated here.

Further, the FlowTracer administrator is expected to be an experienced UNIX user who understands UNIX system processes, UNIX shells, shell scripting techniques, networking administration, and general troubleshooting concepts.

Also in this Guide

FlowTracer Roles

When using FlowTracer in large deployments, three main roles emerge: Administrator, Developer and User. In smaller organizations, a given person may have multiple roles, but the duties of each role still exist.

The FlowTracer Administrator will install and configure the product and help other users to access it. FlowTracer Administrator enables effective use of FlowTracer:

- Installing FlowTracer
- Installing needed licensing
- Configuring FlowTracer for the whole company
- Managing local files
- Establishing security rules
- Defining user roles
- Helping users set up command line environment

A FlowTracer Developer will create and maintain the job definitions and dependency rules for different applications that form the basis for a flow that is registered with FlowTracer to manage.

FlowTracer Developers use the features that define and manage flows:

- Makefiles
- Tcl Scripts
- Flow Description Language (FDL)
- Enabling runtime tracing
    - Encapsulation
    - Instrumentation
    - Interception
- vtk_* Application Programming Interfaces (APIs)
- Command Line Interface (CLI) programs that manage a flow

A FlowTracer User is the recipient of flows pre-developed by a developer. The user relies on the standardization and accuracy provided by FlowTracer to increase their productivity. Typically, they will:

- Execute flows that have been prepared by somebody else.
- Change input files and ask FlowTracer to run the dependent jobs that generate new outputs (this is called "retracing").
- When using the FlowTracer console, the user watches the job nodes changing from purple to yellow to green. When the whole set of nodes is green, the full set of jobs has run and the results can be checked.
- If a job node turns red (FAILED), the user needs to investigate the failure: is it a real failure caused by a problem in the input data, or is it caused by external causes such as disk full, missing licenses, bad machines, dependency violations, or whatever else?
- If the jobs appear to be stuck on some status (think of "color"), the user investigate the cause.

This manual is for topics related to the usage of FlowTracer with pre-existing flows.

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 `.cshrc` csh login script file: `source /<install_path>/<version>/<platform>/etc/vovrc.csh`
`sh, ksh, bash, zsh`	Add the following line to your `.profile` shell login script file: `. /<install_path>/<version>/<platform>/etc/vovrc.sh`

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 Window 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.

User Interfaces

You can access and manage the FlowTracer subsystem in multiple ways. This table explains the four different user interfaces. Mix and match which ones you use.

User Interface	Description	Features
Command Line Interface	The Command Line Interface is the set of programs you can run from the shell command line. These programs form a rich set of operations with command line parameters that provide detailed control of each task. Some tasks such as creating projects can only be done using a CLI command.	Enables automation of work. Commands can be run from a script.
Graphical User Interface	The Graphical User Interface is a custom application client using the client/server architecture. You use this program to have a clean, visual way of accessing the dependency graph, so you can control and manage the tracing of jobs. The graphical interface gives you a convenient way to do ad hoc activities for monitoring and controlling the processing.	Interactive. Offers full and immediate control.
Browser Interface	The browser interface is a web based application built into the FlowTracer system. The FlowTracer system provides the web server and web application as part of the standard installation. The server and application is accessed by a URL to the host. This web application provides a form based interface to access and manage the FlowTracer processing using the browser as your client program. The browser based web application includes links to the documentation.	Easy to use. Part of the normal install.
Tcl	Tcl is the language of choice for writing scripts to manage the FlowTracer system. The FlowTracer system provides a useful set of Tcl extensions that can be used by custom written Tcl scripts to access and manage processing. You can write Tcl scripts that have full control over FlowTracer processing, including the	Programmable, extensible. Most useful to advanced users.

User Interface	Description	Features
	development of new shell command line commands.	

Use FlowTracer Help

FlowTracer documentation is available in HTML and PDF format.

Access the Help when FlowTracer is Running

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

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

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

Projects

Project Registry

The Registry contains the list of all project known to an installation of Altair Accelerator software.

The registry is used by the `vovproject` command, and stores metadata such as the host machine and port number of a project.

```
% vovproject list
    PROJECT                     OWNER  HOST        PORT   STATUS
1   IDT                         johnak linux33     6425   stopped
2   TEM                         johnak mac05       6430   stopped
3   chip1                       johnak maclap      6250   running
4   chiptest                    johnak mac05       6442   obsolete
```

The registry is normally in the directory `$VOVDIR/local/registry` but an alternative location may be specified by setting the environment variable VOV_REGISTRY.

The registry directory contains one entry per project. Each entry is a file with a name of the form *`project@host`*. The file is written in Tcl syntax and contains key information about the project. This is known as a *regentry* file.

As of version 2015.03, the registry places the regentry files for each user's projects under a subdirectory named for their user account. In the following example, the file would be in `$VOV_REGISTRY/cadmgr/vnc@rtda01`.

The `vovproject` command should automatically move a project's regentry file into the user-specific subdirectory the first time the project is started on 2015.03. To revert to an earlier release, you must move the regentry file manually.

For example, an entry called `vnc@rtda01` may contain the following information:

```
# -- This registry file is generated automatically. DO NOT EDIT!
# -- Copyright © 1995-2025, Altair Engineering
set host        "rtda01"
set project     "vnc"
set product     "nc"
set owner       "cadmgr"
set startdate    1331080351; # Tue Mar  6 16:32:31 2012
set version     "2015.03"
set protocol    "8.2.1"
set protocolid  "120150304"
set swd         "/remote/proj/cadmgr/vnc"
set port         6271
set pid          3153
set description {}
set status      "running"
set vovdir      "/remote/release/VOV/2015.03/linux64"
set timestamp    1436228536; # Mon Jul  6 17:22:16 2015
set swdmap(keys)    {unix canonical registry windows full }
set swdmap(unix)            {/remote/proj/cadmgr/vnc}
set swdmap(canonical)       {${VNCSWD}}
set swdmap(registry)        {/remote/proj/cadmgr/vnc}
set swdmap(windows)          {r:/VOV/vnc}
set swdmap(full)             {/remote/proj/cadmgr/vnc}
```

The project's regentry file is initially created by the `vovproject create` command. Thereafter, the running vovserver updates its registry entry file every 10 to 30 minutes. During normal shutdown, the server updates the timestamp to the shutdown time and sets the `status` value to "`stopped`".

To list your project entries in the registry, use the following command:`

```
% vovproject list
```

The above command will only list projects belonging to the user who issues the command. To get detailed information about all projects, use this command:

```
% vovproject list -all -l
```

> **Note:** If the server stops suddenly because it is killed or its host machine crashes, the registry entry will not reflect the correct status of the project. The `vovproject list` command will show '*obsolete*' for projects whose underlying status is `running` but whose timestamp is older than 1800 seconds (1/2 hour).

When your project shows an *obsolete* status, you can get it back into a useful running status by starting it and enabling it.

```
% vovproject start project-name
% vovproject enable project-name
% vovproject list
```

To determine whether a project's server is running is to use the UNIX `ps(1)` command on the machine which hosts the project.

Administrator Concerns about the Registry

Because creating a project also creates the registry entry, the registry directory must be on a writable filesystem with permissions permitting users to create files there.

As the vovserver program updates their registry entry files periodically, the registry directory must be mountable from any machine which hosts a FlowTracer project. The standard registry directory location satisfies this requirement as it is a subdirectory of the registry that contains the Altair Accelerator software.

> **Note:** A filesystem that is exported as read-only does not meet the requirements; it will not work.
>
> Multiple versions of Altair Accelerator software can be set up to access the same set of projects by sharing the registry directory or the entire `local` directory among the versions. To set this up:
>
> - Move the local directory as a sibling to the version directories.
> - Create symbolic links from the local directory of each version to the shared directory.

Missing Registry Entry Files

Sometimes a project's regentry file may be absent or corrupted (for example, truncated to zero because the owner is over quota). Here are two cases: vovserver running, and not running.

- If the vovserver is running, it will re-write the registry file if possible.
- If the vovserver was not running, you cannot start it using `vovproject`, because the regentry file that contained the project metadata is gone.

You can re-create the regentry file by starting vovserver manually as follows, starting vovserver from the command line.

```
+ determine the machine on which to start the vovserver program by
  examining the project's setup.tcl file
+ get a shell as the project owner on that machine
+ change to the parent of the <project>.swd directory
+ enter the project environment by
% ves <project>.swd/setup.tcl
+ start vovserver
% vovserver -jsb $VOV_PROJECT_NAME
```

Project Archive

In case there are many projects that are obsolete but can be used later, those projects can be compressed and stored in a separate directory with `vovproject archive`.

Each project has a registry and a Server Working Directory. Multiple projects can be created using a FlowTracer installation. When a large number of projects exist, `vovproject list` operation may become slow to complete. This is a good time to archive some projects.

A project can be archived as follows:

```
% vovproject archive chip1
vovproject 08/29/2016 23:52:23: message: Archiving chip1.swd ...
vovproject 08/29/2016 23:52:24: message: SUCCESS: archived project chip1: archived
 swd size=14 KB
```

To list projects in the archived directory, use -archive option. In `vovproject list`, the archived project is no longer listed.

```
% vovproject list -archive
    PROJECT                         OWNER   HOST        PORT   STATUS
1   chip1                           johnak  linux33     6425   stopped

% vovproject list
    PROJECT                         OWNER   HOST        PORT   STATUS
1   chiptest                        johnak  mac05       6442   obsolete
```

To revive an archived project, `vovproject unarchive` as in:

```
% vovproject unarchive chip1
vovproject 08/29/2016 23:56:46: message: Unarchiving chip1.swd ...
vovproject 08/29/2016 23:56:46: message: SUCCESS: unarchived project chip1
% vovproject list
```

```
    PROJECT                       OWNER  HOST       PORT  STATUS
1   chiptest                      johnak mac05      6442  obsolete
2   chip1                         johnak linux33    6425  stopped
```

You can see the unarchived project from `vovproject list`.

The archive directory is located `vovserverdir/.archive` by default. Customized directory can be set as the environment variable VOV_ARCHIVE_DIR.

Manage the Shell Prompt

Command	Description
vep	Change the prompt.
veprestore	Restore the original prompt.
VOV_USE_VEP	Environment variable used to disable the use of `vep`

The default VOV prompt, set with the command `vep`, contains the name of the local host, name and host of the current VOV project, the current environment, and the last two components current directory. Example:

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

The effect of this command is purely cosmetic; its purpose is to make the user 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 `veprestore`:

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

The `vep` command is embedded in other Altair Accelerator commands, notably `vovproject` and `ves`. If you prefer your regular prompt to the one set by `vep`, you can disable it by setting the environment variable VOV_USE_VEP to 0 *before* sourcing your `.vovrc` file. Specifically, this is used by the files `std.vov.aliases` and `std.vov.aliases.sh`, which are called by `$VOVDIR/etc/vovsetup.{csh,sh}`.

When disabled, the `vep` alias still exists, but does nothing. The `veprestore` command is defined in either case.

Server Working Directory

When a project is created, a server working directory (`.swd` directory) is created to hold configuration and state information about the project.

Location and Structure

The default location for the server working directory is within the `vovserverdir` directory, that is typically in the user's home directory, for instance, the `~/vov` directory.

During the project create command, it is possible to set the location for the project's server working directory to be a different place. It can be useful to place the project's server working directory within the project's top level directory. This allows for moving all the files needed for a project from one machine to another by taking files from a single branch of the file system.

The following list shows the structure of a project's server working directory (SWD), which contains the server configuration files for the project.

```
.../project.swd/equiv.tcl
               /policy.tcl
               /resources.tcl
               /security.tcl
               /setup.tcl
               /taskers.tcl
               /logs/
               /...etc...
```

Find the SWD Default Location

From the command line, `vovserverdir` can be used to get information about the directory that is the default location for `.swd` directories. This command can also look for files within that file system branch.

```
% vovserverdir -physical
/home/jjj/vov
% vovserverdir -logical
${HOME}/vov
% vovserverdir -p setup.tcl
/home/jjj/vov/project.swd/setup.tcl
```

Files in the Server Working Directory (.swd)

There are several files are located in the project's server working directory.

File	Description
crontab.*HOST*	If it exists, this is the crontable for the project on a specific host. For information, refer to Run Periodic Tasks with vovcrontab.

File	Description
equiv.tcl	This Tcl script defines the rules to generate logical names for files. Use `vovequiv -s` to check the effect of the file.
exclude.tcl	This Tcl script defines the rules to exclude files from the graph. All integrated tools use this file, while the vovserver ignores it.
policy.tcl	This is the main configuration file, which controls the behavior of the vovserver.
resources.tcl	This file describes the resources available for the project.
security.tcl	This file limits access to the vovserver to a defined set of users. This limitation is set up by specifying security rules.
setup.tcl	This Tcl script is used to set up the environment for a project. Typically, this script sets VOV_HOST_NAME and VOV_PROJECT_NAME and other environment variable(s) that are specific to the project.
taskers.tcl	This file describes the taskers to be connected to the vovserver. The tool `vovtaskermgr` uses this file.

Any change to a configuration file becomes visible to the vovserver by executing the following command:

```
% vovproject reread
```

Server Working Directory Subdirectories

The server working directory contains several subdirectories.

Subdirectory	Description
`db/`	Contains the SQL configuration file.
`equiv.caches/`	Contains preprocessed information derived from the `equiv.tcl` file. This is done because such execution may be time consuming. The result, which depends on the host, is cached in this directory. **Note:** The files in this directory should not be modified by the user.

Subdirectory	Description
environments/	A location to store project specific environments. To do so, add a line to the `setup.tcl` file, similar to the following: `setenv VOV_ENV_DIR ...somewhere.../ $env(VOV_PROJECT_NAME).swd/environments`
jobs/	Used by Accelerator to store the logs for all completed jobs.
journals/	Contains the journals for the project. Each journal is a chronological list of all macro events occurred in the design. It is the project leader's responsibility to eliminate old issues of the journals. Recent issues are useful for debugging purposes.
logs/	Contains various logs. The server log is located in `server.YYYY.MM.DD.log`. The messages from the taskers are recorded in files with a name as follows: `tasker.<host>.N.M`. The file `daily.log` contains the activities done periodically each day. The files resources `.YYYY.MM.DD.log` and tools `.YYYY.MM.DD.log` are created by the server.
logs/checkouts	Used by Monitor to store the logs of all completed checkouts.
saved/	Contains compressed copies of the trace database. Use `vovproject backup` to save a copy; use `vovproject restore` if the main trace is corrupted.
scripts/	This directory holds scripts that related to the project, such as the scripts used for the periodic activities.
sq3/	Deprecated as of version 2015.09, the default directory for the SQLite database file.
trace.db/	Contains the trace database in ASCII and compressed format. • `pr`: The Persistent Representation of the trace, in ASCII format. • `pr.gz`: The compressed trace. • `cr`: The Crash-Recovery files.

There are also subdirectories for each of the daemons that can be connected to the vovserver. Following is a list of some of the directories:

Subdirectory	Description
vovpreemptd/	Accelerator
vovnotifyd/	Monitor and Accelerator

Server Working Directory Name

To avoid issues such as looping, exercise caution with the name of the server working directory.

The SWD can be obtained with the utility `vovserverdir`. For example, the full path can be obtained with:

```
% vovserverdir
/user/john/myproject/vovadmin
```

The logical canonical name can be obtained with:

```
% vovserverdir -logical
${TOP}/vovadmin
```

A "circular loop" problem can occur with the name of the SWD, which can be prevented:

- To compute the full path to the SWD, either parse the setup file or know the rules in the equivalence file.
- To find either the setup file or the equivalence file, the full path to the SWD must be known.

To prevent issues from occurring, using the procedure `vtk_swd_get` in the `policy.tcl` file is strongly recommended.

```
# Example of using vtk_set_swd in the policy.tcl file
vtk_set_swd windows "x:/release/admin"
vtk_set_swd unix    /remote/release/rtda/admin
vtk_set_swd unix1   /net/zeus/release/rtda/admin
```

Another method to avoid accessing the server working directory is with a "thin client", which is described in the following section.

Alerts

VOV issues an "alert" when an event requires attention. An alert can range from information that does not require action to an urgent fault that requires immediate action.

Depending on the event that occurs, an alert may require attention from a system administrator.

VOV supports four alert levels, which are defined in the table below.

INFO	Information only, no action required.
WARN	Warning, a limit is about to be reached.
ERROR	A fault in one of the subsytems. Example: a syntax error in one of the configuration files.
URGENT	A major fault that compromises the behavior of the system and requires immediate attention. Examples: a license violation or a disk full condition.

Alerts can be viewed on the browser or the command line interface (CLI). In addition, alerts are stored in log files.

For the administrator, VOV permits two actions with respect to an alert:

- Acknowledge the alert.
- Delete the alert from view.

Note: Every alert is stored in a log file; deleting an alert from viewing does not delete the record of the alert in the log file.

Maximum Number of Alerts

The vovserver keeps up to a defined maximum number of alerts in view. The maximum number is defined by the parameter *alerts.max*. The default value is 50. If the number of alerts exceeds the maximum, the oldest alert with the lowest level is deleted from the view.

Note: The record of the alert is not deleted from the log file.

Manage Alerts

For viewing alerts, the level of the most severe alert is visible in the title bar of the browser user interface and in the VOV GUI. The most recent alerts can be viewed from the command line interface with the following commands:

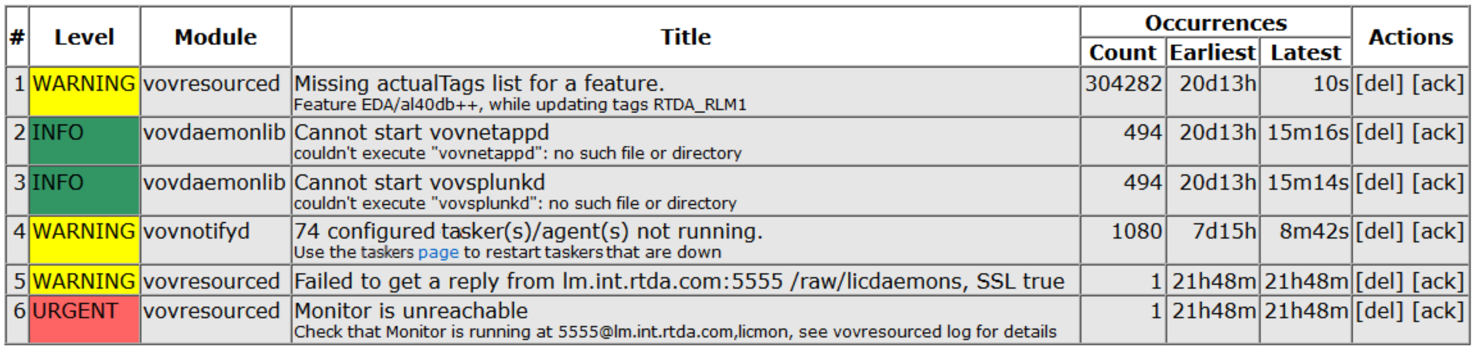
- `vsi` for short format
- `vovshow -alerts` for full format

The following is an example of alerts as shown by `vsi`:

```
Alerts:
```

```
URGENT          Imminent license violation: us      3   12d20h   12d20h
WARNING         License is expiring in less th   2613   12d20h    9d13h
ERROR           License is expiring in less th    558    9d13h    8d13h
URGENT          License has expired               270    8d13h    8d01h
URGENT          License violation                  24    8d13h    8d01h
WARNING         License violation: too many sl    336    8d13h    8d01h
```

The following example shows the same alerts as seen on the browser:



#	Level	Module	Title	Occurrences			Actions
				Count	Earliest	Latest	
1	WARNING	vovresourced	Missing actualTags list for a feature. Feature EDA/al40db++, while updating tags RTDA_RLM1	304282	20d13h	10s	[del] [ack]
2	INFO	vovdaemonlib	Cannot start vovnetappd couldn't execute "vovnetappd": no such file or directory	494	20d13h	15m16s	[del] [ack]
3	INFO	vovdaemonlib	Cannot start vovsplunkd couldn't execute "vovsplunkd": no such file or directory	494	20d13h	15m14s	[del] [ack]
4	WARNING	vovnotifyd	74 configured tasker(s)/agent(s) not running. Use the taskers page to restart taskers that are down	1080	7d15h	8m42s	[del] [ack]
5	WARNING	vovresourced	Failed to get a reply from lm.int.rtda.com:5555 /raw/licdaemons, SSL true	1	21h48m	21h48m	[del] [ack]
6	URGENT	vovresourced	Monitor is unreachable Check that Monitor is running at 5555@lm.int.rtda.com,licmon, see vovresourced log for details	1	21h48m	21h48m	[del] [ack]

Figure 1:

If viewing the documentation from a live vovserver, refer to the Alerts page.

Alerts are also logged in the logs directory in files with names that are formatted as `alert.YYYY.MM.DD.log`. Old alert files are compressed.

Some alerts may not require immediate action. However, it is good practice to acknowledge the alert. The `[ack]` link on the alerts page can be used to indicate that the alert has been acknowledged. The login name of the person acknowledging the alert will be shown on the Alerts page.

Clear Alerts from View

An alert is automatically cleared from view about one day after the last occurrence that triggered the alert. A selected alert can be removed from view by using the `[del]` link from the browser UI.

All alerts can also be cleared from view with the following command:

```
% vovforget -alerts
```

Tcl API

There are two Tcl API procedures in `vovsh` that handle alerts:

- `vtk_generic_get alerts A` - Get alert data into array A
- `vtk_alert_add sev title` - Add an alert

To add an alert from the Tcl interface, use the command `vtk_alert_add`.

To get data for all alerts in Tcl, use the command

```
vtk_generic_get alerts array
```

The following code example shows how `vsi` formats the alerts:

```
# This is how the vsi cmd formats alerts
vtk_generic_get alerts alerts
if { $alerts(count) > 0 } {
    append output "\nAlerts:\n"
    for { set i 0 } { $i < $alerts(count) } { incr i } {
        append output [format "  %-10s %-10s %-30s" $alerts($i,level)
                        [string range $alerts($i,module) 0  9]
                        [string range $alerts($i,title)  0 29] ]
        if { $alerts($i,count) > 1 } {
            append output [format "%7d %8s %8s"
                            $alerts($i,count)
                            [vtk_time_pp [expr $now - $alerts($i,first)]]
                            [vtk_time_pp [expr $now - $alerts($i,last) ]] ]
        }
        append output "\n"
    }
    append output "\n"
}
```

Thin Clients

A *thin client* (VOV protocol) runs on a machine that does not have access to the SWD directory of the vovserver.

Connect a Thin Client

Such a client cannot read the configuration files for the project, such as the `equiv.tcl` file that defines logical filesystem names, or the `setup.tcl` file that defines necessary server environment variables.

An example of a thin client is the `vovinfo` binary, a stripped-down version of `vovsh` that is used for host monitoring in Monitor.

Environment Variables

To use a thin client, it is recommend to set variables as shown below:

```
setenv VOVEQUIV_CACHE_FILE vovcache
setenv VOV_SWD_KEY         none
```

This setting of VOVEQUIV_CACHE_FILE disables checking the `equiv.tcl` file. The client gets equivalences from the server cache, if available, via the VOV protocol RPC. Setting VOV_SWD_KEY to `none` causes the client to avoid accessing the server working directory.

Manage the Server for a Project

Port Number

The project name is normally used to compute the number of the TCP port on which the vovserver listens for connections. This is done by hashing the project name into the port range 6200-6455, using the function `vtk_port_number`.

Clients can connect to the vovserver and interact using either the proprietary VOV protocol or HTTP. It is also possible to connect to a specific "Web" port using either HTTPS or HTTP or to a special read-only port also using HTTP, all of which is explained in Advanced Control of the Product Ports. However, a vovserver may use other than the default port as computed above, when the port is chosen manually at project startup using the `-port` option of the `vovproject create` command, or when there is a conflict with another project that is already using the default port.

VOV normally maps the project name into an integer that ranges from 6200 to 6455 - unless otherwise directed by the environment variable VOV_PORT_NUMBER. This variable is typically set in the `setup.tcl` file.

The following table shows the mapping for some of the standard project names.

Note: `licmon`, the project name for Monitor, uses a non-default port.

Project Name	Port number
intro	6244
licadm	6306
licmon	5555
vnc	6271
wa	6416

To find out a port number that can be used for a project, use `vovbrowser`, or `vsi` in a shell where the project is enabled.

```
% vovproject enable intro
venus intro@mercury [DEFAULT] 201> vovbrowser
http://mercury:6444/project
```

From Tcl, use `vtk_port_number` to compute the default port number for a given project. Example:

```
% vovsh -x 'puts [vtk_port_number vovtutorial]'
6407
```

The option -checkenv in `vtk_port_number` checks the value of the environment variable VOV_PORT_NUMBER.

```
% env VOV_PORT_NUMBER=7777 vovsh -x 'puts [vtk_port_number -checkenv vovtutorial]'
7777
% env VOV_PORT_NUMBER=7777 vovsh -x 'puts [vtk_port_number vovtutorial]'
6407
```

Conflicts in the Port Number

It is possible for different names to map to the same port number, thus making it impossible to run the corresponding servers concurrently on the same host. This occurs because only one process may listen on a host:port at a time, an operating system (OS) restriction.

If this problem occurs, it can be resolved as follows: change the host, change the project name, or explicitly choose the number of the port using the environment variable VOV_PORT_NUMBER. This variable is defined in the `setup.tcl` file in the server configuration directory.

The following subsection describes another method to resolve this issue.

Getting the Port Number in vovsh Connected to vovserver

If writing a script using `vovsh`, and finding the port number on which your script connected to vovserver is needed, the port number can be located by using `vtk_generic_get`. The following example assumes Monitor is owned by "cadmgr" and is using the regular port, 5555.

```
% vovproject enable -u cadmgr licmon% vovsh -x 'vtk_generic_get project P; puts
 $P(port)'
5555
```

This method is recommended, as it queries vovserver for the port number.

From version 2014.11, `vtk_port_number` has the option -checkenv, which causes it to return the value of the environment variable VOV_PORT_NUMBER.

Advanced Control of the Product Ports

Enable a vovserver Project for Command Line Control

The commands described in this chapter require using a "vovproject-enabled shell". This is a command shell that has been configured to work with a specific vovserver. For more information, see Enable CLI Access on UNIX or Enable CLI Access on Windows.

Change the Primary Port on a Running vovserver

The primary port cannot be changed without stopping and restarting the vovserver.

Change the Web Port on a Running vovserver

The web port can be changed dynamically with:

```
% vovsh -x 'VovServerConfig webport 6271'
```

Additional settings related to HTTPS can be set in `policy.tcl`:

```
# Example modification to policy.tcl
set config(ssl.enable)              1
set config(ssl.enableRawUrlOnHttp)  1
```

The vovserver must be notified of changes to `policy.tcl`:

```
% vovproject reread
```

Change the Read-only Port on a Running vovserver

The read-only port, also called the guest access port, can be changed dynamically with:

```
% vovsh -x 'VovServerConfig readonlyport 8283'
```

To maintain backwards compatibility, the read-only port can also be set in `policy.tcl`:

```
# Example modification to policy.tcl
set config(readonly)              8283
```

The vovserver must be notified of changes to `policy.tcl`:

```
% vovproject reread
```

Recommendations for VOV_PORT_NUMBER for Monitor

- Set VOV_WEB_PORT_NUMBER to the well known port 5555
- Set the VOV_PORT_NUMBER to "any"

```
# In licmon.swd/setup.tcl
setenv VOV_PORT_NUMBER     any
setenv VOV_WEB_PORT_NUMBER 5555
```

Recommendations for VOV_PORT_NUMBER for Accelerator

The following is a summary of the important recommendations when setting the VOV_PORT_NUMBER for Accelerator:

- It is important for Accelerator to restart quickly. To do so, avoid conflicts on the port used by the Accelerator vovserver.
- Keep the port for the web pages in the same place even if the vovserver uses a different port after a restart.
- Ensure the web port is different from all other ports. The easiest to do this is to assign the web port the lowest number from the internal list, then increment the value for other ports that you configure.

The following settings for VOV_PORT_NUMBER are recommended:

- Select a series of consecutive values from the internal port list, such as 6271 -6275.
- Set VOV_WEB_PORT_NUMBER to a known port:

For example:

```
setenv VOV_WEB_PORT_NUMBER 6271
```

- If you wish to have anonymous access via HTTP, set VOV_READONLY_PORT_NUMBER to some other fixed value. As mentioned above, our recommendation is +1 above the web port: 6272.
- Set VOV_PORT_NUMBER to a list of 3 or 4 known ports. As mentioned above, select the port numbers by continuing to increment from the value used for the web port.

  For example:

  ```
  setenv VOV_PORT_NUMBER 6273, 6274, 6275
  ```

```
# In setup.tcl for an Accelerator project
setenv VOV_WEB_PORT_NUMBER        6271
setenv VOV_READONLY_PORT_NUMBER   6272
setenv VOV_PORT_NUMBER            6273:6274:6275
```

Recommendations for VOV_PORT_NUMBER for FlowTracer

If you are going to have tens or hundreds of vovservers running on the same small number of machines, then it is most important to avoid conflicts on the ports, a goal that can be achieved by setting the value of VOV_PORT_NUMBER to "any" so vovserver can choose a port.

```
# In setup.tcl for an FlowTracer project
setenv VOV_PORT_NUMBER any
```

Running Multiple Product Instances on the Same Machine

Although not recommended, it is possible to run multiple instances of the same product, such as Monitor or Accelerator, on the same machine. In such cases, it is imperative that you carefully control the allocation of the ports to each instance, in order to avoid collisions (i.e. one of the instances is unable to start) and user confusion.

In particular, you have to make sure that the `setup.tcl` file for each instance in the NC_CONFIG_DIR directory (usually `$VOVDIR/local/vncConfig/<queuename>.tcl` for Accelerator) defines the following environment variables:

- VOV_PORT_NUMBER: Required. This is the main port and it MUST be different for each instance
- VOV_WEB_PORT_NUMBER: If you are using nginx as a front-end for the vovserver, you must assign a distinct value to each instance. Otherwise, you can set this variable to 0 to disable nginx.
- VOV_READONLY_PORT_NUMBER: This is an optional port, which is normally activated for both Monitor and Accelerator. Set this to 0 if you do not need to activate it, or choose a unique value for each instance.

Server Configuration

vovserver configuration parameter values may be changed in a running vovserver using the CLI, or prior to the starting the server via the `policy.tcl` file. An administator can configure the parameters in the running vovserver using the `vovservermgr` command or the `vtk_server_config` procedure.

Note: In usage, all commands and parameters are case insensitive.

Using `vovservermgr`:

```
% nc cmd vovservermgr config PARAMETER_NAME PARAMETER_VALUE
```

Using `vtk_server_config`:

```
% nc cmd vovsh -x 'vtk_server_config PARAMETER_NAME PARAMETER_VALUE'
```

Example of a configuration:

```
% nc cmd vovsh -x 'vtk_server_config timeTolerance 4'
% nc cmd vovsh -x 'vtk_server_config timeTolerance 4'
```

Note: A complete list of the current server configuration parameters is provided in the Server Configuration page.

Server configuration can be controlled by setting variables in the `policy.tcl` file. The variable can be set directly in the "config()" associative array, but it is best to set them with the procedure `VovServerConfig` as in:

```
VovServerConfig VARNAME VALUE
```

In either case, the name of the parameter *`VARNAME`* is case insensitive.

Example

The following is part of the `policy.tcl` file.

```
# This is part of the policy.tcl file.

# Example of parameters set with the procedure VovServerConfig
VovServerConfig readonlyPort  7111
VovServerConfig httpSecure       1

# Example of parameters set by assignment to array config().
set config(timeTolerance)         0

set config(maxBufferSize)        16000000
set config(maxNotifyBufferSize)   400000
set config(maxNotifyClients)      40;
set config(maxNormalClients)      400;

set config(maxAgeRecentJobs)      60;
set config(saveToDiskPeriod)      2h;
set config(autoShutdown)          2w;      # Shut down after 2 weeks of inactivity.
set config(autoLogout)           1h;      # Logout from browser interface.
set config(netInfo)              0;       # Do not collect net information (fs,
 hosts)

# Used by Accelerator for autoforget.
set config(autoForgetValid)      1h
set config(autoForgetFailed)     2d
set config(autoForgetOthers)     2d

set config(autoRescheduleThreshold) 2s

set config(preemptionPeriod) 3s
```

Below is an example of parameters set with the procedure `VovServerConfig`:

```
VovServerConfig readonlyPort  7111
VovServerConfig httpSecure       1
```

Web Server Configuration

HTTP Access Models

There are 3 HTTP access models:

- Legacy
- Internal/External
- Nginx

Legacy Webserver

The Legacy webserver is the basic web server that is internal to vovserver and serves content directly to web browser clients.

All traffic is transmitted using HTTP protocol and is unsecured. This method is approriate for REST versions up to version 2.0.

This is the case when

- `webport=0`, or
- `webport != 0` and `webprovider=nginx`

Internal Webserver

The Internal webserver is an enhanced web server that is internal to vovserver, for secure pages and all REST versions.

The Internal webserver is established when

- `webport != 0` and `webprovider=internal`

To specify the web port at product start, refer to the product-specific documentation for startup. To change the port in an already-running product instance, see *Advanced Control of the Product Ports*. To enable SSL support (HTTPS), follow the steps in Configure the TLS/SSL Protocol.

You get REST v3 API support from this webserver, and we still transparently delegate some HTTP requests to the old web server on the VOV port.

The Internal server securely handles all incoming traffic, decrypting it before handing it off to the locally running vovserver. Likewise, any response that is sent back to the browser is routed through the Internal webserver, which encrypts the response and sends it to the browser. This implementation is known as an SSL termination proxy.

nginx Webserver

The vovserver serves content to a proxy webserver (nginx), which communicates to web browser clients. Under this model, SSL can be enabled, securing all traffic using the HTTPS protocol.

The nginx web server is enabled when the web port is configured with a non-zero value. To specify the web port at product start, refer to the product-specific documentation for startup. To change the port in an already-running product instance, see *Advanced Control of the Product Ports*. To enable SSL support (HTTPS), follow the steps in Configure the TLS/SSL Protocol.

For experts only, advanced customizations to the nginx configuration can be made by modifying its configuration template. Configuration templates are searched for in the following locations:

Order	**Type**	**Path**
1.	Instance-specific	`$SWD/vovnginxd/conf/nginx.conf.template`
2.	Site-wide	`$VOVDIR/local/config/vovnginxd/nginx.conf.template`
3.	Installation-specific(edits not recommended)	`$VOVDIR/etc/config/vovnginxd/nginx.conf.template`

If customizations are intended, it is recommended to start with a copy of the default configuration template shown at location 3 above and place into either location 1 or 2.

Note:

- The configuration template is copied into the nginx configuration directory located at `$SWD/vovnginxd/conf`, named as `nginx.conf`. The copy is made upon product start, as well as any time the web port or SSL configuration is changed.
- Changes to the actual configuration file can be read into nginx via the `vovdaemonmgr reread vovnginxd` command, but such changes will be overwritten the next time the configuration template is copied.
- The configuration template contains keywords surrounded by @ signs, such as @WEBPORT@, that are dynamically substituted with values during the copy process. Removal of these keywords is not recommended, as it may effect the ability for nginx to be reconfigured in the event of a vovserver failover.

Configure the TLS/SSL Protocol

The internal and nginx webservers support TLS/SSL Protocol communication via "https" - prefixed URLs when configured correctly.

Note: TLS encryption is enabled for client communication to the VOV port by default starting in 2025.1.0. TLS can be disabled by setting the new `comm.tls.enable` server configuration parameter to 0. Older version clients that do not use TLS will be allowed to connect without TLS encryption.

The vovserver serves content to a proxy webserver (nginx), which communicates to web browser clients. Under this model, SSL can be enabled, securing all traffic using the HTTP protocol.

When SSL is enabled, nginx will look for an SSL certificate/key pair in the following locations:

Order	Type	Path	Files
1.	Site-wide wildcard	`$VOVDIR/local/ssl`	`wildcard-crt.pem` `wildcard-key.pem`
2.	Host-specific	`$SWD/config/ssl`	`hostname-crt.pem` `hostname-key.pem`
3.	Host-specific (auto-generated and self-signed)	`$SWD/config/ssl`	`hostname-self-crt.pem` `hostname-self-key.pem`

Note:

- For hostname, use the actual host name that will be used to access the web UI. This will be the value of VOV_HOST_HTTP_NAME that was set in the configuration. If not defined, the value of VOV_HOST_NAME is used instead.

  To use the fully qualified domain name, the value of VOV_HOST_HTTP_NAME must be set.
- Self-signed certificates will present security warnings in most browsers.

Updating the TLS/SSL cert requires restarting the webserver so that the cert files can be re-read. For the internal webserver, see, "Restarting the Webserver" below.

Guest Access Port

The vovserver can be configured to enable a guest-access port, also called the read-only port due to the limited privileges allowed by the port. This port bypasses the login prompt and provides the user with a READONLY security principle, which disallows access to writable actions as well as certain pages in the UI.

To specify the guest access port at product start, refer to the product-specific documentation for startup. To change the port in an already-running product instance, follow the steps in *Advanced Control of the Product Ports*.

Transition from nginx Webserver to Internal

To transition from external (nginx) to the internal web server, follow these steps:

1. Shut down nginx with the command `vovdaemonmgr stop vovnginxd`.
2. Delay for 5 seconds with the command `sleep 5`.
3. Start the internal web server with `vovservermgr config webprovider internal`.

Restarting the Webserver

Complete the following steps to restart the webserver without bringing down vovserver.

1. Enter the following:

```
vovservermgr config webport 0
```

2. Wait five seconds, then enter:

```
vovservermgr config webport $VOV_WEB_PORT_NUMBER
```

VOV Protocol Summary

The Altair Accelerator VOV Communication Protocol is TCP-based and is a proprietary implementation that is built on a client-server model. It also uses proprietary encoding and has additional protection against snoop/replay attacks. This protocol has a default timeout of three seconds. To change the default timeout, the parameter, VOV_RELIABLE_TIMEOUT, can be configured.

Two parameters can affect buffers, *maxBufferSize* and *maxNotifyBufferSize* that can be configured in the `policy.tcl` file, which is located in the server working directory (SWD). Following is a summary of the acceptable values for these parameters:

Parameter	**Default**	**Min.**	**Max.**
maxBufferSize	16M	1K	100M
maxNotifyBufferSize	1M	100K	40M

For more information about configurations, please refer to the Server Configuration page.

The primary mechanism that utilizes the VOV protocol is the `vovsh` utility. `vovsh` implements the VTK API and connects to the vovserver. The `vovsh`, and the job wrappers `vw`, `vrt` and `vov` can also be used with the VOV protocol.

About Port Numbers

Altair Accelerator products use distinct ports. Typically, one port is the default listener for the initial vovserver. To compute this port number, the project name is normally used to compute the number of the TCP port on which the vovserver listens for connections. This is done by hashing the project name into the default port range 6200-6455, using the function `vtk_port_number`. The port range can be changed by using the environment variable VOV_PORT_NUMBER

Note: The port range of 6200-6455 is the default range when the VOV_PORT_NUMBER variable is either unset or is set to `automatic`.

The following table shows examples of mapping product names to ports:

Product	**Default Port**	**Use**
Accelerator	6271	Authenticated
Accelerator	6272	Read-Only
Allocator	8787	-
Monitor	5555	Authenticated
Monitor	5556	Read-Only

Product	Default Port	Use
FlowTracer	6200-6455	vovserver Ports

Clients can connect to the vovserver and interact using either the proprietary VOV protocol or HTTP.

Note: Additional information about the VOV protocol is proprietary and is not published. If additional information about the VOV protocol is needed, please contact Altair Engineering Support.

Getting the Port Number with vovsh

If writing a script using `vovsh`, finding the port number to which the script is connected can be located by using `vtk_generic_get`. The following example assumes the Altair Accelerator Monitor is owned by "cadmgr" and is using port 5555.

```
% vovproject enable -u cadmgr licmon
% vovsh -x 'vtk_generic_get project P; puts $P(port)'
55555
```

Conflicts with the Port Number

If different names are mapped to the same port number, the corresponding servers cannot run on the same host at the same time. (This is an operating system restriction, in which only one process can listen on a host:port at a time.)

If this issue occurs, it can be resolved with one of the following actions: change host, change the project name, or explicitly choose the port number by using the environment variable VOV_PORT_NUMBER. This variable is defined in the `setup.tcl` file, which is located in the server configuration directory.

Time Tolerance

The file server (or servers) and the vovserver may be unsynchronized. If that occurs, comparing the timestamp of a file with the start time of a job, may produce an incorrect result due to the clock offset.

While it is easy to synchronize the clocks, it may be occasionally necessary to run VOV on a network that is slightly offset.

Tolerance can be set to accept timestamp comparisons with a tolerance of a few seconds. A typical value for the tolerance is less than 10 seconds. By default, no tolerance is allowed; it is assumed that the network is synchronized.

Setting the Tolerance

Time tolerance is set by adding a line in the `policy.tcl` file. Example:

```
set config(timeTolerance) 2
```

The tolerance can also be set via the browser interface. To do so, log in as ADMIN and visit the Admin page.

Client Limitation and Tuning

The maximum number of clients - the combination of vovtaskers, user interfaces and proxies, that can be concurrently connected to a vovserver is limited by the number of file descriptors available.

This is an operating system parameter, and is inherited from the shell that starts vovserver. It can not be changed after vovserver starts.

There are two kinds of limits, a hard limit and a soft limit. Limits are imposed to reduce the likelihood of exhausting system resources. A soft limit may be set by shell commands so long as a value less than or equal to the hard limit is selected. The hard limits for descriptors and other resources may vary by user, group, and other attributes.

On UNIX, this number is operating system dependent. In most UNIX installations, the hard limit for file descriptors is 1024 or more.

On Windows NT, VOV sets the limit at 256 file descriptors.

On Linux, `root` can change the limits in the file `/etc/security/limits.conf`. Example:

```
* hard nofile 8192
* soft nofile 2048
```

The above example sets the soft limit for all users to 2048, and the hard limit to 8192. The '*' character could be replaced by that of the Accelerator owner account, e.g. 'cadmgr'.

Background

Each operating system offers a limited number of file descriptors for each process. In modern systems, this limit may be up to 65000. The vovserver can handle as many clients as the "descriptors limit" allows. It is also possible to reduce the number by setting a soft limit using the methods described above.

To allow a large number of clients, vovserver must be started with a high limit. The `ncmgr` command reports the number at startup time, please read it carefully before replying 'yes'. Example:

```
% limit descriptors 16000
% ncmgr start
```

The file descriptors are used by the vovserver to communicate with the clients: vovtaskers, GUI, browser, interactive jobs, etc.

The utilization of file descriptors are approximately:

- The server by itself needs about 10
- The other descriptors less than 40 are not used
- Each vovtasker needs 1
- Each running batch job needs 1
- Each running interactive job needs 2
- Each vovconsole needs 2
- Each nc monitor needs 1

Example: On a loaded farm with 500 vovtaskers, each with four CPUs, with half jobs interactive, the estimated number of descriptors needed is:

```
10 + 500 + 500 * 2  + 500 * 2 * 2 =  3510 file descriptors
```

This leaves descriptors for about 580 monitors and GUIs.

Behavior with Exhausted File Descriptors

The exhaustion of file descriptors rarely occurs. Altair Accelerator's main concern is preserving the integrity of the vovserver. Commands that attempt a new connection to the vovserver fail to connect and return an error message too many `clients in the system`. The vovserver will then post an alert showing `too many open files`. In that condition, ordinary commands such as `nc hosts` and `nc list` will not work because the `vovsh` that runs those commands cannot connect to vovserver.

Reserved Connection on the localhost

When file descriptors are exhausted, you can connect to vovserver using a special method through the software loopback interface (lo0, 127.0.0.1). This is achieved by setting VOV_HOST_NAME to localhost. Example:

```
% vovproject enable vncNNNN
% setenv VOV_HOST_NAME localhost
% vsi
```

Solutions to File Descriptor Exhaustion

A short-term solution is to stop some of the clients is to lower the demand for file descriptors. Transient GUI clients such as VovConsole, monitors, and Accelerator GUI should be stopped first. Any idle vovtaskers should also be stopped. Guidelines:

- Check how users are submitting jobs. There are some limits on *`maxNormalClients`* and *`maxNotifyClients`* in the `policy.tcl` file to prevent accidental or malicious denial-of-service attacks. Sometimes we have seen jobs submitted with the -wl option and placed in background, each consuming 2 descriptors.
- Next you should first find whether it is possible to raise the descriptor limit on the current host. If not, a longer-term solution is to move the vovserver to another host that offers more file descriptors. A newer versions of UNIX is a good candidate as they offer 65K descriptors.

It is possible to continue operation, even in the presence of interactive jobs, by moving the vovserver and making the new queue the default queue so that newly-submitted jobs go to the new default queue. The server on the host with limited descriptors will finish all jobs, and it may then be shut down.

Client Service Modes

On Linux-based systems, there are two client servicing modes from which to choose: poll (default) and epoll. The mode chosen specifies which POSIX mechanism the vovserver will use to determine which client file descriptors are ready for use. The mode can be specified by setting `config(useepoll)` to 0 (poll, default) or 1 (epoll) in the `SWD/policy.tcl` file.

Generally, the epoll mode should result in more efficient processing of service requests. As of this version, epoll mode is a new feature and is therefore disabled by default.

Safe Server Threads

As the server farm becomes larger and larger, it becomes useful to allow the vovserver to use multiple threads to handle the costly read-only services, such as listing sets of jobs or retrieving the complete list of taskers.

This capability is controlled by the parameter *`thread.service.max`*, which has a default value of 2. To completely turn off this capability, set the parameter to 0. For example:

```
# In policy.tcl file:  increase number of concurrent threads allowed.
set config(thread.service.max) 8
```

Turn Off Threads

```
# In policy.tcl file:  turn off completely threads.
set config(thread.service.max) 0
```

Recommendation on Linux Systems

Safe threads are forked versions of vovserver, which should live no more than 60 seconds. Typically, these threads need to live for only a few seconds.

If the memory footprint of the vovserver is a significant fraction of the total virtual memory on the job host, the number of threads can be increased by setting the kernel parameter *`vm.overcommit_memory`* to 1.

For example:

```
# Example:
# sysctl -w vm.overcommit_memory=1

# Optional
# sysctl -w vm.overcommit_ratio=50
```

Set the Range for VovId

A VovId is a nine-digit string assigned by VOV to each object in the trace.

Example: "000012345"

The VovIds grow monotonically up to 999,999,999, after which they cycle back down to restart at about 5,000.

Change the Range used for VovIds

The range used for VovIds can be configured from the command line by using the configuration keyword *idrange* and specifying both the low and the high range of the VovIds. There are rules for the relationship between the low and high values for VovId. For example, high must be at least 1,000,000 more than low. These rules are silently enforced.

In the following example, the VovId is constrained to the range of 5 million to 100 million:

```
% vovsh -x 'vtk_server_config idrange 5000000-100000000'
```

The current range of the VovIds can be retrieved with the following command:

```
% vovsh -x 'vtk_generic_get policy a; parray a vovId*
a(vovIdHigh) = 100000000
a(vovIdLow)  = 5000000
```

VovId Changes in FlowTracer (only in versions before 2015.09)

Note: This section does not apply for version 2015.09!

In FlowTracer (but not in Accelerator), the VovId of a job changes every time the job is executed. This happens because a new node is created each time a job starts: a new node is assigned a new VovId. For the duration of the execution, there are two nodes that represent the same job:

- The node with the old VovId and with status *RETRACING
- The node with the new VovId and with status *RUNNING

Upon completion of the job, the new node remains while the old node is forgotten.

The FlowTracer server remembers the relationship between the old VovId and the new VovId; if a job node is requested with the old VovId, the server will retrieve the job with the new VovId.

VovId No Longer Changes in FlowTracer (starting from version 2015.09)

The job that is running on a tasker goes through the status RETRACING=ORANGE to RUNNING=YELLOW but does not change the VovId. The information about which dependencies are new and which ones are old is kept in the "origin" field of the dependencies.

Server Configuration Parameters

Name	Type	Default	Range	Description
acl.default.jobs.	string	VIEW,EXISTS,CHOW		Default ACL for FlowTracer jobs

Name	Type	Default	Range	Description
				assigned to the leader role.
alerts.max	integer	50	[5--1000]	Maximum number of alerts. If more alerts are created, the oldest with the lowest level is archived.
allowcoredump	boolean	0	0,1	If nonzero, request the vovserver to dump core even if it was started from a shell where limitcoredumpsize was zero.
allowForeignJobsO	boolean	0	0/1	Allow jobs from other users to run on non-root taskers. Default=0, set to 1 to allow scheduling of jobs on non-root taskers even if the job belongs to a different user than that of the tasker.
allowUidForSecuri	integer	0	[0--2147483647]	Additional UID that can own the security.tcl file
alm.enable	boolean	0	0/1	Enable Altair Licensing.
autoForgetFailed	time spec	2 days	[5m,infinity]	If the autoForget bit is set on a job, the job will be forgotten automatically upon failure after the specified time.
autoForgetOthers	time spec	2 days	[5m,infinity]	If the autoForget bit is set on a

Name	Type	Default	Range	Description
				job, the job will be forgotten automatically if it is not scheduled after the specified time.
autoForgetRemoveL	boolean	0	0/1	Try to remove the log files associated with auto-forgotten jobs.
autoForgetValid	time spec	1 hour	[5m,infinity]	If the autoForget bit is set on a job, the job will be forgotten automatically upon successful termination after the specified time.
autoLogout	time spec	4 hours	[10m,infinity]	The period of inactivity after which the system automatically logs a user out of the browser interface. The minimum value for this parameter is 10 minutes.
autoRescheduleCou	integer	4	0-10	Specifies the maximum number of reschedule attempts per job.
autoRescheduleOnN	boolean	1	0/1	Controls whether auto-rescheduling will avoid the same host (default) or just the tasker, allowing the job to run on a different tasker on the same host. Does not apply to auto-

Name	Type	Default	Range	Description
				rescheduling based on exit status (see exit-status docs).
autoRescheduleThr	time spec	0s	0s-60s	This parameter deals with jobs that fail quickly. If the failure time is less or equal to the specified value, the job is automatically resubmitted at high priority, with the name of the failing host as a negated resource, so that the job will be placed on a different host (or tasker, configurable) for the next run. If the autoRescheduleThreshold is 0, automatic resubmission is disabled. A maximum of 4 auto-reschedules are permitted by default. This maximum can be controlled via the autoRescheduleCount parameter. Host/tasker behavior can be controlled via the autoRescheduleOnNewHost parameter. Does not apply to auto-rescheduling based on exit status (see exit-status docs).
autostop.timeout		60s		This parameter provides a timeout

Name	Type	Default	Range	Description
				for the execution of all scripts installed in the SWD/ autostop directory. The autostop function is called automatically upon shutdown of the vovserver and all scripts have up to the timeout to executed and exit. This time period is provided to allow autostop scripts to communicate with the vovserver before exiting, while protecting against a problematic script that fails to exit. Once the timeout has been reached, the vovserver continues with its exit procedure, and any autostop scripts that have opened a connection to the vovserver will likely fail. Default can be specified in policy.tcl. Example: `set config(autostop.timeout) 120` This parameter can also be set using the VTK API. Example: `vtk_server_config`

Name	Type	Default	Range	Description
				`autostop.timeout 120`
`blackholeDiscardT`	timespec	10m00s	[2m00s--10h00m]	Failed jobs older than this time are discarded in the blackhole computation
`blackholeFailRate`	double	0.900000	[0.250000--1.00000	Fraction of failed jobs that are running well on another tasker
`blackholeFailedJo`	integer	5	[2--10000]	Number of failed jobs on a tasker to consider the tasker a black hole
`blackholeMaybeTim`	timespec	10s	[0s--1h00m]	Suspension time for tasker that maybe a black hole, but we are not sure yet
`blackholeSuspendT`	timespec	3m00s	[1m00s--10h00m]	Suspension time for a black hole
`bucketValidTasker`	timespec	5m00s	[0s--41d16h]	Control frequency of determining if any valid taskers exist for a bucket. If this parameter is 0, then the facility is turned off.
`cgi.max`	integer	40	[10--200]	Max number of concurrent CGI pages that can be generated.
`checkoutHostLower`	boolean	0	0/1	If set, host name is always forced to lower case in checkouts.

Name	**Type**	**Default**	**Range**	**Description**
`checkoutUserLower`	boolean	0	0/1	If set, user name is always forced to lower case in checkouts.
`clientConnectionQ` *`clientConnectionQ`*				By default, the vovserver responds to multiple incoming connections during the same cycle. This feature can be controlled with these two parameters: *`clientConnectionQueue.mo`* multi or single, default is multi. *`clientConnectionQueue.si`* the size, integer value 25-1024, default is 512.
`comm.buffer.compr`	integer	1	[1--9]	Compression level between 1 and 9. 1 is speed first and 9 is percentage of string data in buffer to trigger buffer compression.
`comm.buffer.compr`	integer	4000	[100--1000000]	Minimum packet size to trigger buffer compression.
`comm.buffer.compr`	integer	50	[1--100]	Minimum percentage of string data in buffer to trigger buffer compression.
`comm.maxBufSize`	unsigned	16777216	[65536--214748364	Maximum size of communication

Name	**Type**	**Default**	**Range**	**Description**
				buffers between server and client.
comm.maxTimeToCom	integer	5000	[1000--20000]	
comm.minBufSize	unsigned	32	[32--2147483647]	Minimum size of communication buffers between server and client.
comm.minTimeForCo	integer	1000	[1000--2000]	
comm.mmapThreshol	unsigned	4294967295	[4096--4294967295	Size at which comm buffers are allocated via mmap.
comm.requireClien	boolean	0	0/1	Require clients authenticate with a vov security key.
comm.tls.enable	boolean	1	0/1	Enable TLS for vov protocol clients.
cpuprogressWindow	integer	1	[1--1440]	Number of samples to average when calculating cpuprogress (allowed range: 1-1440, default value: 1)
crashRecoveryMaxE	timespec	1m00s	[0s--30m00s]	Max extension of crash recovery period due to taskers reconnecting during quiet time.
crashRecoveryPeri	timespec	60s	[30s,5m]	This parameter specifies how long the server is in crash recovery mode. The default value is for 60 seconds after which the serer

Name	Type	Default	Range	Description
				will enter normal operation.
crashRecoveryQuie	timespec	30s	[0s--5m00s]	Possible crash recovery delay due to tasker reconnection during quiet time before crash recovery ends.
dashboard.thresho	integer	300	[1--1000000000]	Minimum number of buckets to mark as "critical" bucket count in the web dashboard. (default: 300)
dashboard.thresho	integer	200	[1--1000000000]	Minimum number of buckets to mark as "high" bucket count in the web dashboard. (default: 200)
dashboard.thresho	integer	50	[1--1000000000]	Minimum number of buckets to mark as "low" bucket count in the web dashboard. (default: 100)
dashboard.thresho	integer	100	[1--1000000000]	Minimum number of buckets to mark as "medium" bucket count in the web dashboard. (default: 100)
dashboard.thresho	double	3.000000	[0.001000--30.0000	Minimum scheduler time to mark as "critical" scheduler activity in the web dashboard, in seconds. (default: 3.0)

Name	Type	Default	Range	Description
dashboard.thresho	double	1.000000	[0.001000--30.0000	Minimum scheduler time to mark as "high" scheduler activity in the web dashboard, in seconds. (default: 1.0)
dashboard.thresho	double	0.100000	[0.001000--30.0000	Minimum scheduler time to mark as "low" scheduler activity in the web dashboard, in seconds. (default: 0.5)
dashboard.thresho	double	0.500000	[0.001000--30.0000	Minimum scheduler time to mark as "medium" scheduler activity in the web dashboard, in seconds. (default: 0.5)
dashboard.thresho	integer	90	[0--100]	Minimum percentage to mark as "critical" server size in the web dashboard, as a percentage of total host RAM. (default: 90)
dashboard.thresho	integer	60	[0--100]	Minimum percentage to mark as "high" server size in the web dashboard, as a percentage of total host RAM. (default: 60)
dashboard.thresho	integer	20	[0--100]	Minimum percentage to mark as "low"

Name	Type	Default	Range	Description
				server size in the web dashboard, as a percentage of total host RAM. (default: 40)
dashboard.thresho	integer	40	[0--100]	Minimum percentage to mark as "medium" server size in the web dashboard, as a percentage of total host RAM. (default: 40)
defaultStopSignal	string	TERM,HUP,INT,KILL		Default stop signal cascade; used for job control.
defaultStopSignal	integer	1	[0--20]	Default stop signal delay; number of seconds to wait between transmitting signals to taskers when using vovstop and related commands. (allowed range: 0-20, default: 1)
defaultSuspendSig	string	STOP		Default suspend signal; used for job control
disablefileaccess	boolean	0		If this parameter is greater than 0 (zero), the server will not access any file that is not under $VOVDIR or $VNCSWD. By 'access' we mean any operation on the file like stat(), open(), access().

Name	Type	Default	Range	Description
				This is a protection to prevent a possible server freeze if it tries to access files on an dead NFS server. If this parameter is greater than 1 (one), then not only the server disables access to files, but also the CGI clients that could possibly show file information will disable such functionality. In summary: • Under normal conditions, set this parameter to 0 • If you are concerned about the vovserver blocking on a bad NFS mount, set this parameter to 1 • If you want maximum information security, set this parameter to 2.
`diskspacecheck.mi`	unsigned	1000	[1--50000]	Specify free disk space alert generation threshold (in MB) for free disk space

Name	Type	Default	Range	Description
				checking (allowed range: 1-50000, default: 1000)
diskspacecheck.mi	integer	1	[1--50]	Specify free disk space alert generation threshold (in %) for free disk space checking (allowed range: 1-50, default: 1)
emptyBucketAge	integer	120	[0--86400]	Delete empty buckets older than the specified age in seconds.
enableBackfillRes	boolean	0	0/1	Enable/Disable backfilling jobs on resource reservation.
enableLdap	boolean	0	0/1	If set, the system uses also LDAP for browser authentication. Available on the following architectures: macosx, linux64, armv7l
enableLookAheadRe	boolean	0	0/1	Enable/Disable job based reservation.
enablePam	boolean	1	0/1	If set, the system uses PAM for browser authentication.
enablePartialRese	boolean	0	0/1	Enable/Disable partial tasker reservation.
enableTimestampCh	boolean	1	0/1	On large traces, it is good to

Name	Type	Default	Range	Description
				completely disable the checking of the timestamps of files.
enableTimestampCh	boolean	1	0/1	On large traces, it is good to disable the checking of the timestamps of the upcone of a target because it may take too long.
enableUserArtifac	boolean	0	0/1	Enable/Disable cleanup of user-created artifacts including fairshare groups, sets, and limit resources that are no longer in use.
enableWaitReasons	boolean	1	0/1	Enable logging of Wait-Reasons.
enterpriselicense	string	Auto		Enterprise licensing mode
enterpriselicense	boolean	0	0/1	Use burst licenses if available.
enterpriselicense	timespec	5m00s	[10s--1h00m]	With enterprise licensing in automatic mode, wait this much before decreasing again.
enterpriselicense	timespec	4s	[1s--5m00s]	With enterprise licensing in automatic mode, wait this much before increasing again.
epollbuf	integer	50	[50--1024]	Epoll buffer size

Name	Type	Default	Range	Description
failover.maxdelay	integer	120	[10--3600]	Specifies the maximum duration during which a vovtasker can participate in a failover server election (default: 120).
failover.usefailo	boolean	0	0/1	If set, only taskers in the failover tasker group participate in the failover server election (default: 0).
fairshare.allowAd	boolean	0	0/1	Allow ADMIN to bypass ACL when modifying a FairShare group.
fairshare.compute	boolean	1	0/1	Enable/disable FairShare computation
fairshare.default	integer	100	[0--100000]	Default weight assigned to new FairShare groups. Can also be controlled by a sibling group called 'default'.
fairshare.default	timespec	1h00m	[0s--7d00h]	Default window assigned to new FairShare groups. Normally the groups inherit the window from their parent of from a sibling group called 'default'.
fairshare.maxjobs	integer	20	[1--1000000]	Max number of jobs to dispatch

Name	Type	Default	Range	Description
				from each bucket at once.
fairshare.maxjobs	integer	20	[1--1000000]	Max number of jobs to dispatch on each loop, after which the FairShare stats will be recomputed.
fairshare.nocompu	timespec	2m00s	[1m00s--1h00m]	If FairShare computation is disabled and FairShare has not been updated externally, maximum time to wait (in seconds) before computing FairShare.
fairshare.oversho	boolean	1	0/1	If set, each bucket will be limited by its max dispatch depth for each scheduler cycle (default: 1).
fairshare.relativ	integer	1	[0--2]	Assuming: t=target allocation; h=historic allocation in window; r=running allocation now. If set to 2, compute FairShare as a weighted sum of d = (r-t) + alpha(h-t). If set to 1, compute FairShare distance relative to the FairShare target d=((r-t)+(h-t))/t (of course assuming t>0). If set to 0,

Name	Type	Default	Range	Description
				compute FairShare distance by a simple difference between actual and target d=(r-t)+(h-t).
fairshare.relativ	double	10.000000	[0.001000--1000.00	Weight of historic distance relative to running distance in computing value for FairShare ranking. Only used if fairshare.relative is set to 2.
fairshare.strictN	integer	0	[0--1]	Perform strict acl checks when creating non-USER type nodes. If set to 0, user only needs ATTACH acl on parent. If set to 1, user needs CREATE acl on parent.
fairshare.updateP	timespec	4s	[0s--1h00m]	Update FairShare stats no more frequently than specified period.
hero.to.monitor.f	integer	30	[1--10000]	How often to send resource usage updates from Hero to Monitor.
hero.to.monitor.u	boolean	0	0/1	Enable resource usage updates from Hero to Monitor
hog.protection.cl	integer	1	[1--600]	Specify the client processing delay (in seconds) for users under hog protection (allowed

Name	Type	Default	Range	Description
				range: 1-600, default: 1).
hog.protection.en	boolean	0	0/1	Enable hog protection in scheduler; note that this is independent of hog.protection.time.enable (default: 0)
hog.protection.jo	integer	100000	[1000--9999999]	Specify currently-managed (all states) job count threshold to trigger hog protection against a user (allowed range: 1000-9999999, default: 100000
hog.protection.ti	double	1.000000	[1.000000--100.000	Multiple of allocated service time required before classifying a user as a hog (allowed range: 1.0-100.0, default: 1.0
hog.protection.ti	double	1.000000	[0.100000--100.000	Multiple of service-time exceeding allocated service time to apply as a delay for service hogs (allowed range: 0.1-100.0, default: 1.0)
hog.protection.ti	boolean	1	0/1	Enable time-based hog protection in scheduler; note that this is independent of hog.protection.enable (default: 1)

Name	**Type**	**Default**	**Range**	**Description**
hog.protection.ti	integer	30	[1--300]	Lookback window when computing time-based service hogs, in seconds (allowed range: 1-300, default: 30)
hog.protection.ti	double	2.000000	[0.000000--100.000	Minimum percentage of real-time taken by a user's services to be considered an "active user" for service hog calculations (allowed range: 0.0-100.0, default: 2.0)
hog.protection.ti	double	40.000000	[0.000000--100.000	Minimum percentage of real-time taken by all services to enable time-based service hog calculation (allowed range: 0.0-100.0, default: 40.0)
http.fileAccessDe	string	ALLOW-OWNER		Whether to ALLOW, ALLOW-OWNER, or DENY file access as the default for REST based file access calls.
http.maxReadOnlyS	string	30d		The maximum time a read-only session can be valid before it expires.
http.maxSSLVersio	string	TLSv1.3		The maximum SSL/TLS version supported by the internal webserver.

Name	Type	Default	Range	Description
http.minSSLVersio	string	TLSv1.3		The minimum SSL/TLS version supported by the internal webserver.
http.proxytimeout	unsigned	60	[5--10000]	HTTP Proxy Timeout (seconds)
httpSecure	boolean	1		This should be set to 1 or "enable" if you use the browser user interface. If it is set to "0", browser users do not have to authenticate, and hence the server does not know their login identity. The default value is 1. When this parameter is 1, the server requires basic authentication to serve most of its pages. By setting this parameter to 0 (zero), any access via the HTTP protocol is allowed, but the server will not know user's login identity. This is provided for backwards compatibility, but the use of the value is discouraged. If this parameter is set to 2, or "disable", or any other number, then

Name	Type	Default	Range	Description
				the HTTP interface is completely disabled.
http.tls12Ciphers	string	AES256-GCM-SHA384:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:AES128-GCM-SHA256		The list of ciphers supported by TLS 1.2 and older when using the internal webserver.
http.tls13Ciphers	string			The list of cipher suites supported by TLS 1.3 when using the internal webserver.
http.workerthread	unsigned	5	[2--50]	HTTP Worker Threads. Change requires vovserver restart.
isProductionSetup	boolean	0	0/1	Specifies whether this queue is deployed in a production environment.
jetstreams.enable	boolean	1	0/1	Enable jetstreams in scheduler (default: 1)
jetstreams.jobLim	integer	100	[0--2147483647]	Limit the number of jobs dispatched per jet stream (allowed range: 0-INT32_MAX, default: 100, 0 = unlimited)
jetstreams.thresh	integer	5	[1--86400]	Specify maximum job duration (in seconds) for utilizing jetstreams (allowed range:

Name	Type	Default	Range	Description
				1-86400, default: 5)
jobprofile.usedb	boolean	0	0/1	If set to 0, save job profile to in-memory wave If set to 1, save job profile to time-series database
keptJobsCleanupCh	integer	10000	[100--1000000]	keepfor jobs cleanup chunk size.
keptJobsCleanupIn	timespec	5m00s	[1m00s--1h00m]	Interval in seconds for kept jobs cleanup.
la.adjustForUncon	boolean	1	0/1	Enable/disable adjust for unconsumed allocations.
la.adjustForUncon	timespec	2m00s	[1m00s--5m00s]	Time window to check for unconsumed allocations for adjustment.
la.feature.expira	timespec	1h00m	[30s--1d00h]	Time threshold since feature data last received from LM, after which LA will stop allocating this expired feature.
la.ooq.waves.disa	boolean	0	0/1	Disable wave based OOQ estimation.
license.AltairUni	boolean	0	0/1	Set to 1 to enable Altair Units licensing.
license.promise		7200	1 minute more than the refresh	Specifies the time, in minutes, past the most recent

Name	Type	Default	Range	Description
			time, but greater than 20 - 7200.	refresh when the RLM license will be checked in. Default: 7200.
license.refresh		60m	10m-1 minute less than the promise time.	Specifies the rate, in minutes, for the vovserver to refresh its RLM license checkout.
liverecorder.logd	string	/tmp		Directory in which the Live Recorder recording file should be saved. The directory must exist. Default is '/ tmp'.
liverecorder.logs	integer	256	[256--65536]	Live Recorder log size in MB (range: 256-65536 default: 256)
lm.lookAndFeel	string	latest		License Monitor 'look and feel' version. Default is 'latest'.
log.nodestatus.ch	boolean	0	0/1	Enable additional detail on node status changes.
log.nodestatus.lo	integer	0	[0--3]	Global nodestatus logging type (0=NONE, 1=ALL, 2=INVALID, 3=DOWNCONE)
logs	boolean	0	0/1	Enable tasker log compression.
logs,checkouts,co	boolean	1		You can control which log files get compressed by editing the policy.tcl file

Name	Type	Default	Range	Description
				and setting the config(log,XXXX,compress) flags.
logs,checkouts,ha	string			A command line to process the log file as it is being created. The command line should contain the variables @FILENAME@ and @EXPIRE@.
logs,jobs,compres	boolean			You can control which log files get compressed by editing the policy.tcl file and setting the `config(log,XXXX,compress` flags.
logs,jobs,handler	string			A command line to process the log file as it is being created. The command line should contain the variables @FILENAME@ and @EXPIRE@.
logs,journals,com	boolean	0		You can control which log files get compressed by editing the policy.tcl file and setting the `config(log,XXXX,compress` flags.
logs,journals,han	string			A command line to process the log file as it is being created.

Name	Type	Default	Range	Description
				The command line should contain the variables @FILENAME@ and @EXPIRE@.
`logs,resources,co`	boolean	0		You can control which log files get compressed by editing the `policy.tcl` file and setting the `config(log,XXXX,compress` flags.
`logs,resources,ha`	string			A command line to process the log file as it is being created. The command line should contain the variables @FILENAME@ and @EXPIRE@.
`logs,server,compr`	boolean	0		You can control which log files get compressed by editing the policy.tcl file and setting the `config(log,XXXX,compress` flags.
`logs,server,handl`	string			A command line to process the log file as it is being created. The command line should contain the variables @FILENAME@ and @EXPIRE@.
`logs,tasker,compr`	boolen	0	0/1	You can control which log files

Name	Type	Default	Range	Description
				get compressed by editing the policy.tcl file and setting the `config(log,XXXX,compress` flags.
`logs,waitreason,c`	boolean	0		You can control which log files get compressed by editing the policy.tcl file and setting the `config(log,XXXX,compress` flags.
`logs,waitreason,h`	string			A command line to process the log file as it is being created. The command line should contain the variables @FILENAME@ and @EXPIRE@.
`longServiceWarnin`	double	0.800000	[0.000000--1000000	Warn if a service takes a long time, as defined by this parameter (in seconds).
`lookAheadSchedInt`	timespec	1s	[0s--1m00s]	Interval in seconds for lookahead scheduling.
`lookAheadTimeReso`	timespec	1m00s	[1s--1h00m]	Interval of discrete time instances at which available resource count is maintained, in seconds.
`maxAgeRecentJobs`	timespec	60	[0,3600]	This defines the max age of the jobs in the set System:transitionsRecent,

Name	Type	Default	Range	Description
				which contains all the jobs that have recently finished. Due to various scheduling delays, it is possible for some jobs in the set to be somewhat older.
`maxBufferSize`	unsigned	16777216	[65536--214748364	Maximum size of communication buffers between server and client. The same with comm.maxBufSize.
`maxChildren`	integer	20		Maximum number of children managed by vovserver. This mostly affects the number of concurrent CGI requests that the server can manage.
`maxCommandLineLen`	integer	40960	[128,100000]	Number of characters in the longest command line that will be accepted by the system. A job containing a longer command line will be rejected.
`maxEnvLength`	integer	512	[128,16000]	Maximum length allowed for the environment field in a job. Example: If the environment is BASE+GNU, length of the string is 8.

Name	Type	Default	Range	Description
maxJobArray	integer	10000	[10,100000]	This parameter defines the maximum size of a job array.
maxLevel	integer	4095	[3,4095]	Maximum number of levels in the dependency graph. Dependency declarations that cause the levels in the graph to exceed this value are rejected.
maxLookAheadReser	integer	10	[1--1000]	Maximum lookahead reservation count at a given time.
maxLookAheadSched	timespec	4h00m	[1h00m--1d00h]	Maximum lookahead reservation scheduling timeout in hour.
maxNormalClients	integer	400		Maximum number of normal clients per user. This is a protection against denial-of-service attacks by a single user.
maxNotifyBufferSi	integer	1000000	[100k,40M]	In bytes, this parameter controls the size of the buffers used internally to notify asynchronous clients. If you see "overflow" events and are annoyed by it, you can increase the size of the buffer.

Name	Type	Default	Range	Description
maxNotifyClients	integer	40		Maximum number of notify clients per user. This is a protection against denial-of-service attacks by a single user. The 'notify' clients are those that tap into the event stream. They are used for example in waiting for jobs to finish and to update the GUI.
maxORsInResourceE	integer	20	[0--25]	Max number of OR operators in a resource expression.
maxPathLength	integer	1024	[128,16000]	Maximum length of names of files, including the end of line terminator. This is different from all other "max" parameters for jobs. This means that if this parameter has value 1024, then the actual max length of a path is actually 1024-1=1023.
maxResMap	integer	5000	[1, infinity]	Maximum resources map number that can be set in the resource map set.

Name	**Type**	**Default**	**Range**	**Description**
maxResourcesLengt	integer	1024	[128,16000]	Maximum length of the resources field for a job.
maxSmartSets	integer	50	[10--5000]	Max number of smart sets.
maxeventqueue	integer	100000000	[1000--500000000]	Max number of events in event queue.
metrics.autoSaveP	timespec	5m00s	[1m00s--1h00m]	Period for auto-saving metrics data files to disk.
metrics.enable	integer	7	[0--7]	Enable the metrics system. The old name 'enableMetrics' can also be used. The value is a mask, with the following bit values: 0=disable_all_metrics 1=enable_server_metrics 2=enable_design_metrics 4=enable_fairshare_metrics 7=enable_all_metrics
metrics.keepFiles	timespec	30d00h	[0s--3y00d]	How long to keep saved metrics data files.
metrics.logdir	string	/tmp		Directory in which the metrics file should be saved. The directory must exist. Default is tmp.
metrics.maxmem	integer	2000	[100--100000]	Max size in MB for each metrics family. If the memory exceeds this amount, the system drops some

Name	Type	Default	Range	Description
				metrics or some points.
metrics.usedb	boolean	0	0/1	If set to 0, save metrics to in-memory metrics database If set to 1, save metrics to time-series database.
minMemFreeOnServe	integer	10	[0--10000]	Issue alert if free memory on server goes below this minimum threshold, in MB.
minSwapFreeOnServ	integer	10	[0--10000]	Issue alert if free swap on server goes below this minimum threshold, in MB.
minhw	string	"RAM/20 CORES/1 SLOTS/1 PERCENT/1"		The minimum resources requested by a job.
mm.fork	boolean	0	0/1	Enable use of MADV_DONTFORK for client buffers.
mm.huge	boolean	0	0/1	Enable Huge Pages
nfsdelay	timespec	0s	[0s--5m00s]	Do not even check files that have changed less than 'nfsdelay' ago.
notifyMaxEffort	integer	20	[5--50]	Percent of time to give to notification of clients
notifySkip	integer	0	[0--100000]	OBSOLETE: NO LONGER USED In inner loop, do not notify clients for

Name	Type	Default	Range	Description
				each event, but skip a few.
prSaveTimeout	timespec	1h00m	[1m00s--41d16h]	Time (seconds) after which an incomplete PR save operation exits.Default is 1 hour
preemption.load	boolean	1	0/1	Load/unload preemption module
preemption.log.al	boolean	0	0/1	Enable debug messages for all preempt rules
preemption.log.ve	integer	0	[0--10]	Control verbosity of log messages for preemption.
preemption.max.ti	double	0.300000	[0.001000--10.0000	Max time spent in preemption for each loop
preemption.max.ti	double	0.300000	[0.001000--10.0000	Max time spent in preemption for each rule. Any rule that exceeds this will be disabled.
preemption.module	string	libpreemption.so		Preemption module name
preemption.rule.c	time spec	"1y"	[0,"1y"]	Specifies how long rules are disabled due to exceeding preemption.max.time.rule. 1y = Indefinite, 0 = generate alert only. Default is 1y (Indefinite).
preemptionFairsha	timestamp	0		If greater than the current date, enable computation

Name	Type	Default	Range	Description
				of statistics for FairShare of preemption. Normally set by vovpreemptd. For testing, use vtk_server_config "preemptionFairshare" "on", which activates the computation for the next 10 minutes.
preemptionPeriod	time spec	3s	[0,1000h]	This parameter specifies how often Accelerator will attempt to perform preemption. The default value is 3 seconds which will cause preemption to be attempted every 3 seconds. If a value of 0 is specified then this will disable preemption.
preemptionResourc	time spec	90s	[0,1000h]	This parameter controls the frequency of releasing features for suspended jobs as a result of preemption. If this parameter is 0, the code is never called.
processRetentionT	timespec	5s	[0s--5m00s]	How long a process record (see procinfo feature) is kept in memory since the last update.

Name	Type	Default	Range	Description
prop.maxNameSize	unsigned	255	[16--1024]	Max length of name a property
prop.maxStringSiz	unsigned	131071	[128--1048576]	Max length of value of a string property. Any value longer than this will be truncated.
query.processor.s	boolean	0	0/1	If set to 1, use the stream processor (faster) for vovselect/ vtk_select_loop queries, when feasible and supported by the client
rds.enable	boolean	0	0/1	Enable LM-to-NC license resource updates over sockets instead of via vovresourced. Default: false
rds.secure	boolean	0	0/1	Enable security on the RDS event port. Default: false
rdsMaxEffort	integer	20	[5--80]	Percent of time limit for updates from RDS
readonlyport	integer	0	[0--65535]	The readonly port for vovserver
rejectIndirectTas	boolean	0	0/1	Reject jobs submitted through the binary indirect tasker
resMapGroupDefaul	integer	50	-1 (unlimited)	This parameter sets the default value for the resource map "Group:groupname"

Name	Type	Default	Range	Description
				for a new group. This resource map limits the maximum number of jobs that the group can run at the same time. For each individual group, you can control the value of the map with a `vtk_resourcemap_set Group:groupname number` statement in the `resources.tcl` file. A negative value of this parameter is interpreted as "unlimited."
resMapToolDefault	integer		-1 (unlimited)	This parameter sets the default value for the resource map "Tool:toolname" for a new tool. This resource map limits concurrent number of jobs using the tool. For each individual tool, you can control the value of the map with a `vtk_resourcemap_set Tool:toolname number` statement in the `resources.tcl` file. A negative value of this parameter is

Name	Type	Default	Range	Description
				interpreted as "unlimited."
resmap.max.map.le	integer	1024	[512--64000]	Max length of the map field in a resourcemap
resmap.max.maps.i	integer	300	10, 50000	Max number of resourcemaps considered in the evaluation of a resource expression. This is a protection against cycles in the resource maps (e.g. a b c). The typical value is less or equal to the total number of resource maps in the system.
resmap.max.nru.ti	double	0.100000	[0.000100--1.00000	Max time allowed to compute NRU handles
resmap.sw.types	string	License Limit Policy User Group Priority		Comma or Space separated list of types of resources that are guaranteed to be resource maps (i.e. SW resources), even if they are not currently defined as a resource map. This parameter is typically an empty list in Accelerator Plus and includes 'License Limit Policy ...' for the other products.

Name	Type	Default	Range	Description
resources.usedb	boolean	0	0/1	If set to 0, use resources data from *.SWD/data to generate wave files If set to 1, use resources data from time-series database to generate wave files
restApiMaxFileSiz	integer	1000	[10--10000]	REST Api max file size in KB.
resuserDisableMat	integer	1000	[0--1000000]	Threshold of LM handles and FT jobs to match. If the sum of LM handles and FT jobs exceeds this threshold, do not perform matching.
resuserEnableNRUM	boolean	1	0/1	Matching check should include checking for Not-Requested-Used licenses.
resusermatchtoler	timespec	0s	[0s--10m00s]	Tolerance (in seconds) allowed when matching jobs and handles
resusermaxmatches	integer	6	[0--40]	Maximum number of 'also' matches between jobs and lm handles
runAsWX	integer	0	[0--1]	Set up this project to run as Accelerator Plus.
sanity,smartsets	boolean	0	0/1	Enable sanity check of smart sets
sched.busy.thresh	integer	3	[2--1000000]	If the delay between poll()

Name	Type	Default	Range	Description
				is more than this threshold, then the server is considered to be 'busy'
sched.maxpostpone	integer	10000	[1--1000000]	Exit dispatch loop after this number of jobs have been postponed, indicating a loaded or busy farm
sched.megapoll.th	integer	100	[5--10000000]	Threshold that defines how many clients are needed to declare that we are in a 'mega-poll' situation
sched.taskerstats	boolean	0	0/1	Enable scheduler statistics for taskers (used in unusedhw.cgi)
schedMaxEffort	integer	20	5 50	If the scheduler becomes expensive, i.e. if it takes more than 0.02 seconds to dispatch jobs, the vovserver adaptively skips some scheduling cycle to give time to the clients to interact with the vovserver. In this way, the scheduler effectively changes from a pure event-driven behavior to a cycle-based behavior. The cycle is controlled by the schedMaxEffort

Name	Type	Default	Range	Description
				variable, which sets the ratio between the effort spent in the scheduler and the effort spend elsewhere. Example: If the schedMaxEffort is 20 (meaning 20% of time should be spent in the scheduler) and the scheduler time is 0.1s, then the scheduler will be called no more frequently than once every 0.5 seconds (0.1*100/20=0.5)
sds.enable	boolean	0	0/1	Enable/disable Streaming Data Service
seatlic_max	integer	2147483647	[0--2147483647]	Override the maximum number of seat licenses that can be checked out (default: unlimited).
setQueueEnv	boolean	-	0/1	Whether to set NC_QUEUE in the job env to Accelerator Plus project.
skipConflictingJo	boolean	0	0/1	Enable/Disable processing of subsequent jobs when the top job in the bucket is conflicting with

Name	Type	Default	Range	Description
				the future tasker reservation
ssl.enable	boolean	1	0/1	Enable SSL for web server
ssl.enableRawUrlO	boolean	1	0/1	Enable /raw url's for SSL
substituteArrayId	boolean	0	0/1	Substitute references to @ARRAYID@ in resource string with array ID (default: 0). Caution should be used because this will result in a job queue bucket being created for each submission of a job array.
substituteArrayIn	integer	0	[0--2]	Substitute references to @INDEX@ in resource string with array index (default: 0). Caution should be used because this will result in a job queue bucket being created for each job in an array. If set to value of 2 then find and reuse existing buckets.
substituteJobIdIn	boolean	0	0/1	Substitute references to @JOBID@ and @IDINT@ in resource string with job ID (default: 0).

Name	Type	Default	Range	Description
				Caution should be used because this will result in a job queue bucket being created for each job.
switchToExtBufThr	integer	100000	[10k,1...31]	In bytes, this parameter controls the size of the internal buffer used for vtk_set_list_elements_enhar If the size exceeds this parameter, then vovserver switches to an external buffer.
taskIdleReqTime	timespec	2m00s	[0s--7d00h]	Required task idle time to run the user-created artifacts cleanup.
tasker.attachDeta	integer	-1	[-1--1]	Reserved for internal use only.
tasker.authorizat	integer	5	[0--60]	Minimum time in seconds that vovserver waits before authorizing new taskers
tasker.childProce	boolean	0	0/1	If TRUE, taskers should kill all of a job's child processes when that job exits (i.e. assume child processes running after jobs exit are zombie processes); default is FALSE.
tasker.childProce	string			If set in conjunction with tasker.childProcessCleanup,

Name	Type	Default	Range	Description
				then taskers should kill all of a job's child processes when that job exits, except for those named here in a comma separated list. default is ""
tasker.max.reserv	integer	10	[0--100]	Maximum number of reservations per tasker
tasker.maxWaitToR	timespec	4d00h	[5s--1y00d]	Time to wait after losing connection to server before reconnecting again.
tasker.minWaitToR	timespec	5s	[1s--1y00d]	Min time to wait after losing connection to server before reconnecting again.
tasker.props.enab	boolean	1	0/1	Enable saving of message properties for all taskers
tasker.slotLimit	integer	1000	[1--2147483647]	Maximum number of slots per tasker
tasker.uninterrup	boolean	0	0/1	If TRUE, taskers will ignore incoming job control requests if an existing request is in-process; default is FALSE.
taskerBusyUponDis	boolean	1	0/1	Specifies whether taskers should be set to busy upon job dispatch

Name	Type	Default	Range	Description
taskerDisconnResv	timespec	5m00s	[1m00s--1d00h]	Delay in cleaning up the tasker reservation on tasker connection error
taskerheartbeat	integer	60	[1--300]	Default heartbeat for taskers, in seconds. Can be overridden for individual taskers via vovtaskermgr configure or by starting the vovtasker with the -U argument
taskerload.usedb	boolean	0	0/1	If set to 0, use taskerload data from *.SWD/data to generate wave files If set to 1, use taskerload data from time-series database to generate wave files
tasksMaxEffort	integer	20	[5--50]	Percent of time to give to maintenance tasks
thread.preempt.ma	integer	40	[10--200]	Max number of concurrent threads for auxiliary preemption work like vovlmremove
thread.service.en	boolean	1	0/1	Enable safe threads for VovFsGroups
thread.service.en	boolean	1	0/1	Enable safe threads for queries
thread.service.fo	double	0.020000	[0.001000--2.00000	Reference cost for a fork() used to

Name	**Type**	**Default**	**Range**	**Description**
				decide when to fork() for service threads
thread.service.ma	integer	0	[0--50]	Max number of concurrent forked threads for read-only services
thread.service.mi	integer	50000	[0--40000000]	Min Number of nodes in set to activate safe threads
thread.service.mi	integer	500	[0--1000000]	Min Number of resmaps to activate safe threads
thread.service.mi	integer	500	[0--100000]	Min Number of taskers to activate safe threads
timeTolerance	time spec	0	[0,600]	Control the time tolerance of the server. The default value of 0 may be too strict for some networks. You will usually need to have ntp running to synchronize the clocks to use zero, but this is the best practice. Values up to about 3 seconds will not impose much performance penalty.
trustHostReported	boolean	0	0/1	Used mostly for demos
trustUserReported	boolean	0	0/1	Check for client user inconsistencies

Name	Type	Default	Range	Description
unsetSkipFlagOnCo	timespec	30s	[1s--24m00s]	Interval in minutes to unset the skip flag on job conflicting with the future reservation
useepoll	boolean	0	0/1	Switch between poll(0, default) or epoll(1) for client servicing (Linux only).
userArtifactClean	timespec	4h00m	[1h00m--1d00h]	Maximum timeout for forcefully cleaning up user-created artifacts that are no longer in use, irrespective of server load
userArtifactClean	timespec	10m00s	[1m00s--1d00h]	Minimum timeout for cleaning up user-created artifacts when the server is under low load or idle
userGroupLowerCas	boolean	0	0/1	Force user group name to be lower case
userLowerCase	boolean	0		If set to 1, force all user names to lowercase. This applies also to ACL and to the security modules (vtk_security).
usrtmp	path	/tmp		/tmp, /usr/tmp, /var/tmp, c:/temp
vovIdHigh	integer	999,999,999	[1,000,000 - 2,000,000,000]	When recycling vovid's, this is the largest usable ID. This will be at least

Name	Type	Default	Range	Description
				1,000,000 more than vovIdLow.
vovIdLow	integer	5,000	[5,000,10,000,000]	When recycling vovid's, this is the ID that the that generator wraps back to.
vovstring.maxLeng	unsigned	2147483647	[65535--429496729	Max length of a VovString object
vovwxd.arraysizep	boolean	0	0/1	If set, the server will set a property for the final launcher array size when using local resources
vovwxd.fastexit	boolean	1	0/1	Enable/disable vovwxd fastexit
vovwxd.localresou	boolean	0	0/1	If set, the user facing queue will manage local resourcemaps
waitreasons.usedb	boolean	0	0/1	If set to 0, use waitreasons data from *.SWD/data to generate wave files If set to 1, use waitreasons data from time-series database to generate wave files
webport	integer	0	[0--65535]	The web port used by http server (possibly with ssl)
webprovider	string	internal		The facility hosting the web port. Either internal or nginx.

Name	Type	Default	Range	Description
wx.dispwithbucket	boolean	1	0/1	Use Bucket:XXX resource shortcut in Accelerator Plus
wx.setQueueEnv	boolean	0	0/1	Whether to set NC_QUEUE in the job env to Accelerator Plus project.

Troubleshooting the Server

If the server does not start, take the following steps:

1. Check if the server for your project is already running on the same machine.

   Note: Do not start a VOV project server more than once.

   For example:

   ```
   % vovproject enable project
   % vsi
   ```

2. Check if the server is trying to use a port number that is already used by another vovserver or another application. VOV computes the port number in the range [6200,6455] by hashing the project name. If necessary, select another project name, or change host, or use the variable VOV_PORT_NUMBER to specify an known unused port number. The best place to set this variable is in the `setup.tcl` file for the project.
3. Check if the server is trying to use an inactive port number that cannot be bound. This can happen when an application or the server terminates without closing all its sockets.
4. When a port is not available, the server will exit with a message similar to the following:

```
...more output from vovserver...
vs52 Nov 02 17:34:55          0                3      /home/john/vov
vs52 Nov 02 17:34:55 Adding licadm@venus to notification manager
vs52 Nov 02 17:34:55 Socket address 6437 (net=6437)
vs52 ERROR Nov 02 17:34:55 Binding TCP socket: retrying 3
vs52 Nov 02 17:34:55 Forcing reuse...
vs52 ERROR Nov 02 17:34:58 Binding TCP socket: retrying 2
vs52 Nov 02 17:34:58 Forcing reuse...
vs52 ERROR Nov 02 17:35:01 Binding TCP socket: retrying 1
vs52 Nov 02 17:35:01 Forcing reuse...
vs52 ERROR Nov 02 17:35:04 Binding TCP socket: retrying 0
vs52 Nov 02 17:35:04 Forcing reuse...
vs52 ERROR Nov 02 17:35:04
PROBLEM:  The TCP/IP port with address   6437   is already being used.
```

```
POSSIBLE EXPLANATION:
        - A VovServer is already running   (please check)
        - The old server is dead but some
          of its old clients are still alive  (common)
        - Another application is using the
          address  (unlikely)

ACTION: Do you want to force the reuse of
        the address?
```

In this case, do the following:

a) List all VOV processes that may be running on the server host and that may still be using the port. For example, you can use:

```
% /usr/ucb/ps auxww | grep vov
john   3732  0.2  1.5 2340 1876 pts/13   S 17:36:18  0:00 vovproxy -p
 acprose -f - -b
john   3727  0.1  2.2 4816 2752 pts/13   S 17:36:16  0:01 vovsh -t /rtda/
VOV/5.4.7/sun5/tcl/vtcl/vovresourced.tcl -p acprose
...
```

b) Wait for the process to die on its own, or kill the process with the command `vovkill`. For example:

```
% vovkill pid
```

c) Restart the server.

If there is trouble connecting to the vovserver, check the canonical name of the server working directory. For information, refer to Server Working Directory.

Change the Project Name, Server Host, or User

Change the Project Name

To change the project name, change the variable VOV_PROJECT_NAME in the `setup.tcl` file.

The following example uses a C-shell:

```
% cd `vovserverdir`
% vovproject stop oldname
% mv oldname.swd newname.swd
% vi newname.swd/setup.tcl            # Change VOV_PROJECT_NAME
% vcs newname.swd/setup.tcl.
% vovproject start
```

Change the Server Host

To change the host of a project, the vovserver must be stopped and the host references fixed in the following files:

- The `setup.tcl` file in the server working directory
- The registry entry (normally located in `$VOVDIR/local/registry/user/`)
- For Accelerator only, the queue-specific configuration file in the NC_CONFIG_DIR directory (usually `$VOVDIR/local/vncConfig`)

1. Change to the server working directory.
   a) If the server is still running:
      ```
      % cd `vovserverdir`
      ```
   b) If the server is not running, find the directory:
      ```
      % vovproject list -l | grep projectName
      ```
2. If necessary, shut down the server as shown below.
   ```
   % vovproject stop projectName
   ```
3. In the server working directory, edit the `setup.tcl` file and change the value on the line that sets the environment variable VOV_HOST_NAME.
4. Log onto the new host.
5. Restart the project on the new host:
   ```
   % vovproject start projectName
   ```
6. Clean up the registry. In the directory `$VOVDIR/local/registry/user/`, locate the file `projectName@oldHostName` which can be safely removed. There is another registry entry `projectName@newHostName` which can be kept.

Rehost a Project

To rehost a project, start the project on a different host and pass the -rehost option.
For example:

```
% vovproject start -rehost
```

Block a Project from Returning to Shell

The option -block is used to prevent the project startup from returning to the shell. This is useful when the project is being started as a batch job in a scheduler environment. The -block option can be used when starting or creating a vovproject.

Examples:

```
% vovproject start -block
% vovproject create -block
```

Change Ownership of VOV Projects

It can be necessary or useful to change the owner of a VOV project. For example, a project that was started under an individual's account, but will be shared and should be owned by a role account instead.

VOV records the initial creator of a project as the owner of the project in the project's regentry file. This enables the `vovproject list` command to show a user the projects that the user has created.

There are several ways change the ownership of a project. Regardless of the method used, it is **essential** to first stop the project's vovserver before making any changes. The vovserver periodically updates the status of a project and re-writes the regentry file; any changes made while the project is running may be overwritten by the vovserver.

The following are two examples for changing project ownership.

Change Ownership, Method 1

1. Run `vovproject stop`, and then run `vovproject destroy`.
   This will remove the project's `.swd` and registry entry.
2. Log in as the desired user.
3. Recreate the project's `.swe` and registry entry
   This destroys all record of the existence of the project, including the state of the jobs and files in the project's flow, which may be undesirable.

Change Ownership, Method 2

In this example, you'll stop the project, change the OS ownership of its files, and manually change the project metadata recorded by VOV in the regentry file. Follow the steps to do so:

1. Locate the project's configuration (.swd) directory with the command `vovproject list -all -l`.
   The `.swd` directory is in the directory shown in last column.
2. Change they ownership using OS commands, such as `chown -R` on UNIX or Linux.
3. Edit the `security.tcl` file in the `.swd` to add the new owner as admin.
4. Save the file so that it is owned by the new owner, and is not readable by anyone else.
5. Locate the regentry file in the registry, being careful to get the one for the right VOV version if you are not in a `local` directory that is shared among versions.
6. Open the regentry file and locate the line that looks like:

   ```
   set owner        "previous-owner"
   ```

7. Change the name between the quotes to the desired login account write and quit.
8. Use `vovproject list -all` to verify it shows the desired owner for the project.
9. Get a shell as the desired owner on the project's vovserver host, and start the project using `vovproject start project-name`
10. Carefully review the result for messages about ownership.

Log Files

The server generates several daily log and data files, including:

- Server log files in `*.swd/logs`
- Alert log files in `*.swd/logs`
- Journal files in `*.swd/journals`
- Jobs files in `*.swd/data/jobs`
- Resources files in `*.swd/data/resources`
- Taskerload files in `*.swd/data/taskerload`
- Waitreasons files in `*.swd/data/waitreasons`
- Checkouts files in `*.swd/data/checkouts` (this applies only to Monitor)
- Denials files in `*.swd/data/denials` (this applies only to Monitor)

The files are closed after midnight and a new file is opened for the next day with the appropriate name.

It can be desirable to process the log files as they are being created. The log handler is a program that reads the log as it is being created. Typically, the log handler is invoked by substituting two keywords in the command line for the handler:

- `@LOGFILE@` is the full path of the log file.
- `@EXPIRE@` is the expiration timestamp for the log file, typically midnight of the current day.

A useful log handler is included in VOV, called `vovloghandler`. The usage of the log handler:

```
vovloghandler: This script is an example of a log handler to be used with dailylogs
vovloghandler: It is used to start a process to monitor the logs
vovloghandler: and to process them as they get generated.
vovloghandler: In this example, we use a simple 'tail -f ...'
vovloghandler:
vovloghandler: Usage: vovloghandler @FILENAME@ @EXPIRE@
vovloghandler:
```

The log handler is executed in the background and it must be able to exit by itself. Examples of conditions to terminate a log handler:

- The expiration time has been reached.
- A file called `@FILENAME@.closed`; `@FILENAME@` is substituted with the full path to the log file.

```
#
# This is a fragment of policy.tcl
#
# Example of flags to control which handlers are called for the log files:
#
set config(logs,server,handler)       ""
set config(logs,journals,handler)     ""
set config(logs,jobs,handler)         "vovloghandler @FILENAME@ @EXPIRE@"
set config(logs,resources,handler)    ""
set config(logs,waitreasons,handler)  ""
set config(logs,checkouts,handler)    "vovloghandler @FILENAME@ @EXPIRE@"
```

Log files can be checked by using the script `logchecker.py`. This returns extensive amount of help when called with `--man` or `--usage` flags such as:

```
#/usr/bin/python2.7 ./src/scripts/aeware/rtda/logchecker.py <--man or --usage>
```

Old logs can be cleaned up using `vovcleanup`.

Compress Log Files

After closing a daily log file, the server can automatically compress the file to save on disk space. The utility to compress the file is `vovcompresslog`, which calls gzip or any other utility to compress the files.

```
vovcompresslog: Usage: vovcompresslog <LOGNAME>
```

To control which log files are compressed, edit the `policy.tcl` file and set the `config(log,XXXX,compress)` flags. Example:

```
#
# This is a fragment of policy.tcl
#
# Example of flags to control which logs get compressed.
#
set config(logs,server,compress)      0
set config(logs,journals,compress)    1
set config(logs,jobs,compress)        0
set config(logs,resources,compress)   1
set config(logs,waitreasons,compress) 0
set config(logs,checkouts,compress)   1
```

Reopen the Logs After an Error

If the logging encounters an error, such as a full disk, logging will be suspended and an alert will be issued. After clearing the full disk condition, restart the logs with the following command:

```
% vovsh -x 'vtk_server_config reopenlogs 1'
```

Autostart Directory

With the command `vovautostart`, on vovserver startup, scripts can be specified to execute automatically.

In UNIX, the scripts can be written in either C-shell or Tcl syntax.

Note: For the script to work in Windows, Tcl syntax must be used. Guidelines follow:

- Create a directory named `autostart` in the server working directory.
- For both UNIX and Windows:
  - Create a script with the suffix `.tcl` in the `autostart` directory.
- For UNIX only, CSH scripts are also supported:

- Create a script with the suffix `.csh` in the `autostart` directory.
- Ensure the script has the appropriate executable permissions.

Each script in the `autostart` directory is called with one argument, which is the word *`start`*. This argument is usually ignored in OEM scripts, but can be used to in custom scripts to enforce different behaviors between a manual call on the CLI versus an automated call by the vovserver.

Examples are available in the directory `$VOVDIR/etc/autostart`.

vovautostart

The scripts are launched by the utility `vovautostart`. To repeat the execution of the `autostart` scripts, `vovautostart` can be executed from the command line.

```
vovautostart: Usage Message

  DESCRIPTION:
      Execute the scripts in the *.swd/autostart directory.
      There are three types of scripts that get executed:
      1.  Scripts that match *.sh are executed directly (Unix only)
      2.  Scripts that match *.csh are executed directly (Unix only)
      2.  Scripts that match *.tcl are executed by vovsh.

      The scripts are executed in alphabetical order in the background, with a 5s
      a 5s delay between successive scripts.

      This utility is normally invoked by vovserver upon launching.

  USAGE:
      % vovautostart [optional directory spec]

  EXAMPLES:
      % vovautostart
```

Autostop Directory

With the command `vovautostop`, on vovserver shutdown, scripts can be specified to execute automatically.

In UNIX, the scripts can be written in either C-shell or Tcl syntax.

Note: For the script to work in Windows, Tcl syntax must be used. Guidelines follow:

- Create a directory named `autostop` in the server working directory.
- For both UNIX and Windows:
    - Create a script with the suffix `.tcl` in the `autostop` directory.
- For UNIX only, CSH scripts are also supported:
    - Create a script with the suffix `.csh` in the `autostop` directory.
    - Ensure the script has the appropriate executable permissions.

Each script in the `autostop` directory is called with one argument, which is the word `stop`. This argument is usually ignored in OEM scripts, but can be used to in custom scripts to enforce different behaviors between a manual call on the CLI versus an automated call by the vovserver.

Examples are available in the directory `$VOVDIR/etc/autostop`.

The vovautostop Command

The scripts are launched by the utility `vovautostop`. To repeat the execution of the `autostop` scripts, `vovautostop` can be executed from the command line.

```
vovautostop: Usage Message

  DESCRIPTION:
      Execute the scripts in the *.swd/autostop directory.
      There are two types of scripts that get executed:
      1.  Scripts that match *.sh are executed directly (Unix only)
      2.  Scripts that match *.csh are executed directly (Unix only)
      2.  Scripts that match *.tcl are executed by vovsh.

      The scripts are executed in alphabetical order,
      in the background, and
      with a 5s delay between successive scripts.

      This utility is normally invoked by vovserver upon shutdown.

  USAGE:
      % vovautostop [OPTIONS] [optional directory spec]

  OPTIONS:
     -v            Increase verbosity.
     -h            This usage message.

  EXAMPLES:
      % vovautostop
```

Run Periodic Tasks

Run Periodic Tasks with vovliveness

If the directory "tasks" exist in the Server Working Directory, the server calls the `vovliveness` script once per minute.

The script executes all the tasks contained in the "tasks" directory.

```
vovliveness: Usage Message

  DESCRIPTION:
      This script is called by vovserver about once a minute.
      It can be used to perform maintenance tasks.

  USAGE:
      % vovliveness [OPTIONS] <taskdirectory> <timestamp>

  WHERE:
     task_directory     -- is the directory with the tasks
                           to be executed. The tasks are those
                           that match the expression "live_*.tcl".
     timestamp          -- Currently ignored.

  OPTIONS:
     -v                 -- Increase verbosity.
```

There are many uses for `vovliveness`. Examples are available in the directory `$VOVDIR/etc/liveness`.

To activate this functionality, create the directory `tasks` and add some tasks files with a name matching the expression `live_*.tcl`. The Tcl interpreter has access to all `vtk_*` procedures. Example:

```
% cd `vovserverdir -p .`
% mkdir tasks
% cd tasks
% cp $VOVDIR/etc/liveness/live_start_taskers.tcl .
```

Following an example of the script `live_start_taskers.tcl` to restart any down taskers, once per hour:

```
#
# Copyright © 2007-2025, Altair Engineering
#
# All Rights Reserved.
#
# Directory : src/scripts/liveness
# File      : live_start_taskers.tcl
# Content   : Start down taskers once an hour.
# Note      :
#
# $Id: //vov/branches/2019.01/src/scripts/liveness/live_start_taskers.tcl#3 $
#

set now [clock seconds]

# Get or initialize period
if { [catch {set period [vtk_prop_get 1 LIVE_START_TASKERS_PERIOD]}] } {
    set period 3600
    catch {vtk_prop_set 1 LIVE_START_TASKERS_PERIOD $period}
}
```

```
# Get age
if { [catch {set lastRun [vtk_prop_get 1 LIVE_START_TASKERS_LAST]}] } {
    set lastRun 0
}
set age [expr {$now - $lastRun}]

if { $age >= $period } {

    # Start down taskers
    if { [catch {exec vovtaskermgr start >&@ stdout} errmsg] } {
        VovError "Failed to start taskers: $errmsg"
    }

    # Reset the last run TS
    catch {vtk_prop_set 1 LIVE_START_TASKERS_LAST $now}

}
```

Alerts from Liveness Tasks

Alerts may occur that are related to liveness tasks such as "The previous liveness script is still connected", especially in Monitor.

Note: In previous releases, there is no control these occurrences; such occurrences cause no harm.

The liveness tasks system is designed to support short jobs that are triggered frequently (about once per minute) by the vovserver so long as it is running. It was also used for the database loading task for Monitor checkouts and Accelerator jobs; sometimes these jobs run significantly longer.

In later releases the debuglog parsing, batch reports and other maintenance items are converted to periodic jobs that run on a dedicated vovtasker named 'maintainer', so these alerts should no longer appear.

Run Periodic Tasks with vovcrontab

The UNIX utility `crontab` is used to perform regularly scheduled tasks such as retracing an entire project each night or storing a back-up of the trace every Saturday. `vovcrontab` is a VOV utility that simplifies the creation of cron rules for a project. Directions are provided in this section.

Usage: voncontrab

```
vovcrontab: DESCRIPTION:
vovcrontab:     Interface to the UNIX utility crontab.
vovcrontab:
vovcrontab: USAGE:
vovcrontab:     % vovcrontab [option]
vovcrontab:
vovcrontab: OPTIONS:
vovcrontab:   -help        -- Get this message
vovcrontab:   -new         -- Install the crontab for this project
vovcrontab:                   Also used to update the scripts/vovdir.csh script.
vovcrontab:   -noautostart -- Do not install autostart script to update
vovcrontab:                   vovdir.csh; the default is to install it.
```

```
vovcrontab:   -reinstall   -- Reinstall current crontab for this project
vovcrontab:   -clear       -- Clear the current crontab
vovcrontab:   -show        -- Show the crontab
vovcrontab:   -type <type> -- Specify project type
vovcrontab:
vovcrontab: NOTE:
vovcrontab:     Please remember to copy
vovcrontab:     $(VOVDIR)/etc/autostart/update_crontab_vovdir.csh
vovcrontab:     into your autostart directory if needed.
vovcrontab:     It is installed by default.
```

Enable a Project

Enable a project in a shell via:

```
'vovproject enable <PROJECT>'
```

Create crontabs

Execute `'vovcrontab -new'` to create crontabs.

```
% vovcrontab -new
vovcrontab: Creating vnc.swd/scripts/cron.csh
vovcrontab: Creating cron table vnc.swd/crontab.lion
no crontab for john
vovcrontab: Installing new crontab.
vovcrontab: Installing updated crontab
```

This program prepares the scripts `$SWD/scripts/cron.csh` and `$SWD/crontab.hostname`.

Display Current crontab Definition

Running `vovcrontab -show` shows the current crontab definition.

```
% vovcrontab -show
#### (vovcrontab) START PROJECT vnc ####
#
# ... some lines omitted...
#
# Every hour at 5 minutes before the hour.
55 * * * *  /home/john/vov/vnc.swd/scripts/cron.csh hourly
#
# Every day: at 23:15
15 23 * * *     /home/john/vov/vnc.swd/scripts/cron.csh daily
#
# Every week: on Saturday at 7:00am
0  7  * * 6     /home/john/vov/vnc.swd/scripts/cron.csh weekly
#
# Every month: on the 1st at 3:00am
0  3  1 * *     /home/john/vov/vnc.swd/scripts/cron.csh monthly
#### (vovcrontab) END PROJECT vnc ####
```

Customize the crontab

The crontab can be customized by editing either `$SWD/crontab.hostname` or `$SWD/scripts/cron.csh`. Afterwards, `vovcrontab -reinstall` will need to be run to take the modifications into consideration.

```
% vovcrontab -reinstall
vovcrontab: vnc.swd/scripts/cron.csh  exists already.
```

```
vovcrontab: vnc.swd/crontab.lion exists already.
vovcrontab: Installing new crontab.
vovcrontab: Installing updated crontab
```

Delete the Current crontab Definition

To delete current crontab definitions, use:

```
'vovcrontab -clear'
vovcrontab: vnc.swd/scripts/cron.csh  exists already.
vovcrontab: vnc.swd/crontab.lion exists already.
vovcrontab: Removing the crontab
```

Complete the Cleanup

To complete the cleanup, remove the `crontab.HOSTNAME` file in the SWD directory of the project.

```
% rm crontab.lion
```

Job Status Triggers

The daemon `vovtriggerd` taps the event stream and executes commands that are based on selected events.

A typical application is updating an external SQL database when a job is completed.

Triggers are different from post-commands. Triggers are executed by `vovtriggerd`, which is normally run by the user who owns vovserver. The owner of the account that was used to start vovserver is the *owner* of vovserver. Post-commands are executed by the user who owns each job.

The following table summarizes the information about `vovtriggerd`:

Config file	`vnc.swd/vovtriggerd/config.tcl`
Sample config file	`$VOVDIR/etc/config/vovtriggerd/config.tcl`
Info file	`vnc.swd/vovtriggerd/info.tcl`

Set Up vovtriggerd

`vovtriggerd` is a daemon that is configured as follows:

- Create a subdirectory called `vovtriggerd` in the server configuration directory
- Create a configuration file called `config.tcl` with the main purpose of overriding the procedure called `triggerCallBack`
- Start the daemon:

```
% mkdir `vovserverdir -p vovtriggerd`
% cd `vovserverdir -p vovtriggerd`
% mkdir autostart
% cp $VOVDIR/etc/config/vovtriggerd/config.tcl .
% vovdaemonmgr start vovtriggerd
```

The TRIGGER Property

The default trigger handler looks for the property `TRIGGER` attached to the object mentioned in the event. If the property exists, it is assumed to be the name of a trigger procedure to be called. There are three arguments for the trigger procedure: `id`, `subject`, `verb`.

The trigger procedures are defined in the `config.tcl` file. Following are the guidelines for implementing a trigger:

- The trigger is stateless.
- The trigger is fast; it should complete within a few seconds.

Following is an example of using the `TRIGGER` property:

- Create a trigger call-back in the `config.tcl` file. In the following example, it is named `trigShowJobEventCB`.

```
#
# This goes in PROJ.swd/vovtriggerd/config.tcl
#
proc trigShowJobEventCB { id subject verb } {
    puts "TrigShowJobEventCB: Just got the event $id $subject $verb"
}
```

- Attach the property TRIGGER to jobs in the flow. Example:

```
% vovprop set -text 000123456 TRIGGER trigShowJobEventCB
% vovprop set -text 000234567 TRIGGER trigShowJobEventCB
```

Trigger Events

Following are the events that are processed by `vovtriggerd`:

- JOBID "JOB" "DISPATCH", when the job is dispatched to a tasker.
- JOBID "JOB" "ERROR", if the job fails.
- JOBID "JOB" "STOP", if the job succeeds.

Handling the OVERFLOW Event

If the `vovtriggerd` daemon receives an overflow event (the verb is the string `OVERFLOW`), the procedure `overflowCallBack` is called with no arguments. The overflow event is an indication of a buffer overflow inside the vovserver, which is typically caused by `vovtriggerd` being too slow in processing the events. Depending on the situation, it may be useful to reinitialize the trigger callbacks.

Example of Submission of Jobs with Triggers

The following example applies to Accelerator:

A trigger can be submitted by setting the `TRIGGER` property. Knowing the name of the trigger callback routine to call is required. In the following example, the name of the trigger callback is `updateDbCallBack`.

```
% nc run -P TRIGGER=updateDbCallBack sleep 10
```

```
# Example of updateDbCallBack
# This procedure is defined in *.swd/vovtriggerd/config.tcl
# Here we update a table called "mytable" based on the
# value of a property called MYPROP.
proc updateDbCallBack { jobid subject verb } {
    switch $verb {
        "STOP" - "ERROR" {
            if [catch {set value [vtk_prop_get $jobid "MYPROP"]}] {
                set value -1
            }

            set    stmt "INSERT INTO mytable (id,value)"
            append stmt " VALUES ( $jobid, $value)"
            VovSQL::init
            set handle [VovSQL::open]
            VovSQL::query $handle $stmt
            VovSQL::close $handle
        }
    }
}
```

Security

The VOV security model consists of assigning a *Security Code* to each client or client group (VovUser Group) and granting the permission to execute critical tasks only to clients with the appropriate security level.

Refer to VovUserGroups for more information about how VovUser Groups are created and maintained.

Security Levels

There are five different security levels.

The security levels are:

Security Name	Numeric Level	Description
ANYBODY	1	No privilege to run any CLI command.
READONLY	2	Minimum privileges; a user can only browse the information but cannot change anything
USER	3	A user can only execute established flows and view non critical information. In particular, a USER: • May create, modify or forget his own jobs • May create, modify or forget his own files • May create, modify or forget his own set • May modify or forget his dependencies • Create/modify/start/stop/forget own tasker • May not forget jobs owned by other users • May not modify jobs owned by other users • May not start and stop taskers • May not stop the server
LEADER	4	Intermediate privileges; a user can create and execute arbitrary flows and view all non-security related information. A LEADER: • May start or stop his own taskers • May forget all jobs, including jobs owned by another user • May save trace DB to disk

Security Name	Numeric Level	Description
		This level is rarely used.
ADMIN	5	Administrator privileges; a user has access to most security information An ADMIN: • May forget jobs owned by other users • May stop jobs owned by other users • May not modify jobs owned by another user (no user can modify another user's jobs) • May stop the server • May stop/modify/forget the taskers • May refresh tasker cache • May destroy user • May destroy host • May create or destroy alerts • May create, modify or destroy resource map • May reserve resource • May create, modify or destroy preemption rule • May create, modify or destroy multi-queue objects (Monitor sites, Accelerator queues, resources) • May create, modify or destroy Monitor objects (licdaemons, features) • May not modify jobs owned by another user (because no user can modify another user's jobs)

The Security File

The file `security.tcl` in the server working directory specifies the security policies. This file must be owned by the project owner and must have read/write access only for the owner.

To change the security file of an active server, use the `vovproject reread` command to make the changes effective.

The security policies are defined by the Tcl procedure `vtk_security`. A summary of the Tcl procedure follows:

```
vtk_security username|-group vovusergroupsecurityLevelhostNameOrIpRange ...
```

where:

`username`
: The login name of a user or + to mean "anybody" or - to mean "nobody".

`vovusergroup`
: The name of a VovUserGroups.

`securityLevel`
: USER, LEADER or ADMIN (case insensitive)

`hostListOrIpRange`
: For an IP range, it must have the form "x.x.x.x-y.y.y.y" (example `192.168.10.220-192.168.10.240`). If it is not an IP range, it is either the name of a host or + to indicate "any host" or - to mean "no host".

Placing the order of the rules in this file is not important. The rules are automatically reordered from the most user specific to the least user specific and from the most liberal to the most restrictive with respect to the security level.

Example: least restrictive security

The least restrictive security grants every user full access from any host.

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

Example: most restrictive security

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

Example: typical case

The following example shows a typical security file, in which different privileges are granted to different users. Note the use of variables in the following example:

```
set servers      { reno milano }
set allhosts     { 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 bob     ADMIN   192.168.0.30-192.168.0.100
vtk_security -group mygroup USER    $allHosts
```

In the example, `mary` is an administrator from any host, and `dan` is an administrator only from `reno` and `milano`. The user `pat` is a LEADER from her machine `elko`, and `fred` has USER privileges with the four machines that are defined in the variable `$allhosts`. Additionally, members of the VovUserGroup "mygroup" have USER privileges from `$allHosts`.

Find the Security Level

To find the security level, use `vtk_user_security` from a Tcl script.

In a CGI environment, the variable SECURITY_LEVEL contains the numeric level of security. Example:

```
% vovsh -x "puts [vtk_user_security]"
5 ADMIN
% env SECURITY_LEVEL=3 vovsh -x "puts [vtk_user_security]"
3 USER
```

Operations by Security Level

The VOV security model consists in assigning a *Security Code* to each client (browser, GUI, CLI) and to grant the permission to execute tasks only to clients with appropriate security level.

VOV defines four ordered privilege levels, shown from the least privileged to the most privileged:

- `READONLY`
- `USER`
- `LEADER`
- `ADMIN`

VOV security is enforced by the server process. Each time that a client requests a transaction, the security level of that client's owner is compared to the definitions in the project's `security.tcl` file, and permission is granted or denied accordingly.

The following table shows the operations permitted to clients according to their privilege level. 'Y' indicates the operation is permitted, otherwise it is not.

VOV Operations allowed by Privilege Level

			Privilege Level			
Object	**Operation**	**Description**	**READONLY**	**USER**	**LEADER**	**ADMIN**
Trace						
	ViewStatus	View status information about jobs and files	Y	Y	Y	Y
	CreateJob	Add job to flow your own		Y	Y	Y
Job						
	RunJobSelf	Run a job you own		Y	Y	Y

VOV Operations allowed by Privilege Level						
			Privilege Level			
Object	**Operation**	**Description**	**READONLY**	**USER**	**LEADER**	**ADMIN**
	RunJobOther	Run a job owned by another		Y	Y	Y
	StopJobSelf	Stop a job you own		Y	Y	Y
	StopJobOther	Stop job owned by another			Y	Y
	ModifyJobSelf	Modify a job you own		Y	Y	Y
	ModifyJobOthe	Modify anything other than legal exit status or resources of job owned by another				
	ModifyJobExit(	Modify legal exit status for job owned by another				Y
	ModifyJobResc	Modify resource requested by job owned by another				Y
	ForgetJobSelf	Forget a job you own		Y	Y	Y
	ForgetJobOthe	Forget a job owned by another			Y	Y
Project						

VOV Operations allowed by Privilege Level						
			Privilege Level			
Object	**Operation**	**Description**	**READONLY**	**USER**	**LEADER**	**ADMIN**
	StartTasker	Start a vovtasker of this project			Y	Y
	StopTasker	Stop a vovtasker of this project		Y	Y	Y
	StartServer	Start this project's server				Y
	StopServer	Stop this project's server				Y
	ViewSecurity	See project security info				Y
	ModSecurity	Change project security info				Y

Protection from DoS Attacks

The vovserver implements several protections against denial-of-service (DoS) attacks. One of these is the detection of clients that use a relatively large share of the server time. These clients are considered *service hogs*, and their subsequent service requests can be ignored for up to 3 seconds.

The statistics for server time usage are reset every 10 minutes.

The only evidence of this protection taking place is information in the server log file.The information would look similar to the following:

Service hog: user john 3.2 > 3.0 longest=0.9 penalty=2

The penalty is the number of seconds that the next service will be delayed.

Configure Security

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

Locate the Security Configuration File: security.tcl

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

Example: 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. See the VovUserGroups page for more information on how to set up and administer these groups.

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

Example: Most Restrictive

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

Example: 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 allhosts     { 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
```

VovUserGroups

VovUserGroups are an access control mechanism. They are internally-defined groups of users that are utilized to make the job of allocating access easier within Altair Accelerator tools.

VovUserGroups by themselves do not grant or restrict access. Once the group names are defined, however, they can be assigned security roles and thus make the job of administering security and access less time consuming.

The `vovusergroup` utility is used to manage VovUserGroups from the command line. This includes creating and deleting groups, appending them, and displaying group members.

VovUserGroups Utilization

VovUserGroups can be effectively utilized in a number of areas throughout the Altair Accelerator product suite. In all Altair Accelerator products, they can be used to easily set up access by mapping a VovUserGroup to a defined security role in the `security.tcl` file that resides in the Server Working Directory (SWD). VovUserGroups can also be used to assign access or restrictions to various components of the web interface.

Usage of `vovusergroup` utility:

```
vovusergroup: Usage Message


  Utility to show, create and maintain "vovusergroup" structures based on
  user lists, Unix groups and LDAP groups.

  USAGE:

     vovusergroup <ACTION> <GROUP_NAME> <OPTIONS>

  ACTIONS:

   populate       Creates a new vovusergroup and populates it from a provided
                  userList, an existing unix group or LDAP group.
                  Any pre-existing vovusergroup with the same name is lost.

   addusers       Creates a new vovusergroup (if necessary) and appends the
                  provided userList.
   removeusers    Removes the specified "userList" list of users from
                  an existing vovusergroup.

   delete         Removes a vovusergroup definition entirely, along with
                  its list of users.
   show           Displays the list of users in an existing vovusergroup.
   list           Displays the defined vovusergroup names

  SYNTAX:

    vovusergroup populate    <groupName> <userList>
    vovusergroup populate    <groupName> -unix <unixGroup>
    vovusergroup populate    <groupName> -ldap <ldapGroup>

    vovusergroup addusers    <groupName> <userList>
    vovusergroup removeusers <groupName> <userList>

    vovusergroup delete      <groupName>
```

```
     vovusergroup show        <groupName>

   ABOUT USER LISTS:

      A "userList" above consists of usernames separated by spaces.
      To generate a vovusergroup from LDAP, you must first have LDAP
      connectivity enabled (see Altair Monitor Administration Guide)

   EXAMPLES:

     vovusergroup populate chipA -ldap chipA_govusers
     vovusergroup addusers myBlk jimc terryw suep ronaldb
     vovusergroup show myBlk
```

VovUserGroups Support in Altair Accelerator Products

Currently, support for VovUserGroups is provided in the `security.tcl` file (in all products), and the `web.cfg` file (in Monitor only).

Taskers

A tasker is a VOV client that provides computing resources, specifically CPU cycles, to the vovserver.

The list of taskers connected to a project is described in the file taskers.tcl, and the main utility to start and stop taskers is `vovtaskermgr`. Additional configuration can be specified with the `taskerClass.table` file.

Types of vovtasker Binaries

The main tasker client is called `vovtasker` but there are other variations of it:

- `vovtasker` can run jobs for more than one user; the success depends on file permissions.
- `vovtaskerroot` has the ability of switching user identity.

  Note: Accelerator is the only Altair Accelerator product that needs `vovtaskerroot`.

- vovtasker.exe has the ability of impersonating users on Windows, subject to a set of rules explained in *Vov Windows Impersonation*.
- `vovagent` is a temporary vovtasker that terminates upon a set of timeouts, and is used mostly in conjunction with LSF or SGE.
- `ftlm_agent` is a "thin-client" version of a vovtasker and can be used by Monitor to start and stop license daemons on remote machines.

  In the following sections, you will learn:

  - The format of the taskers.tcl file, which describes the list of vovtaskers that connect to a project.
  - The attributes of a vovtasker.
  - How to write a taskerClass.table file.
  - How to monitor the vovtaskers.
  - How to start and stop a single vovtasker.

Tasker Compatibility

When a job becomes eligible to run, VOV places it on the fastest vovtasker host which offers all the resources requested when the job was submitted. The Tasker Compatibility page in the web browser UI shows details of a job's compatibility with each vovtasker machine. You can use this page to understand why a job seems stuck.

Access the Tasker Compatibility Page

The Tasker Compatibility page is specific to a particular job. You may access this page from two places:

- The *more* link next to a job on the Job Queue Buckets page. This is the rightmost column, and has a magnifying glass icon.
- The *Tasker Compatibility* link on the Job Details page.

The Job Queue Buckets page is the panel that is displayed when the Queue Buckets tab is selected in the VOV Monitor window.

Tasker Compatibility Page Description

The page consists of a table, with a row for each vovtasker machine. The column headings are:

#	Row number
Tasker	vovtasker name, link to details page of that machine
Info	Compatibility with this job
Missing Resources	Additional resources needed to enable this job to run

There is a button at the page bottom which returns you to the Job Queue Buckets page.

Info Field Meanings

The page consists of a table, with a row for each vovtasker machine.

Busy	Machine is responding to a server message
Full	All job slots have running jobs
Suspended	No jobs may run; not in available time band, or being used interactively
Overloaded	Not accepting new jobs; load average exceeds MAXLOAD in config file
Sick	Server has not received timely heartbeat
Paused	Tasker has been paused by an external signal
Not Connected	Tasker object is not yet connected to a vovtasker process
Tasker Mismatch	Resources provided by this tasker do not match resources requested by the job
Exhausted Resource	All of the resource are in use
Resource Cycle	Resource map entries form a loop
Insufficient time left	When using Offhours, expected job duration would extend into offline period
Unknown	Unspecified reason

The taskers.tcl File

The `taskers.tcl` file describes the taskers for a project.

The `taskers.tcl` file is a Tcl script based on the procedure `vtk_tasker_define`. The synopsis for this procedure:

```
vtk_tasker_define hostname [options]
```

The two following examples both declare three taskers on the hosts apple orange and pear:

```
# Fragment of taskers.tcl file
vtk_tasker_define apple
vtk_tasker_define orange
vtk_tasker_define pear
```

```
# Fragment of taskers.tcl file
foreach host {apple orange pear} {
    vtk_tasker_define $host
}
```

The following procedure supports many options to define the characteristics of the tasker. The options include "-resources <string>" to set the resource list offered by a tasker and -CPUS n to define the number of CPUs in a machine. In the following example, the tasker on apple is set up to offer the resource "big_memory":

```
# Fragment of tasker file
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
```

The default value for all options can be changed with the following procedure `vtk_tasker_set_defaults`, as shown below:

```
# Fragment of taskers.tcl file
vtk_tasker_set_default -resources "@STD@ big_memory" -CPUS 2
vtk_tasker_define apple
vtk_tasker_define orange
vtk_tasker_define    -CPUS 4
```

vtk_tasker_define

There are many options that can be used with vtk_tasker_define and vtk_tasker_set_default.

Option	Argument	Description
-capabilities	`string`	The capabilities that the tasker has. This results in license checkout attempts for the specified capabilities. Possible values: FULL, NC, PROCINFO,

Option	Argument	Description
		NETINFO, EXEC, RT. FULL includes all capabilities. NC contains EXEC, PROCINFO and NETINFO capabilities. Altair Accelerator enables runtime tracing for FlowTracer. Default is FULL.
-capacity	`int`	The number of concurrent jobs (job slots) that the vovtasker can handle.
-cpus	`int`	The number of CPUs in the machine, without affecting maxload and capacity. On most platforms, the number of CPUs is computed automatically.
-CPUS	`int`	Convenience option, equivalent to setting -cpus, -maxload and -capacity at the same time. If *N* is the number of CPUs, this options sets -cpus to *N*, -maxload to *N+0.5* and -capacity to *N*.
-coeff	`double`	The tasker coefficient. It is used as a divisor in computing the effective power of a vovtasker, e.g. a coefficient of 2.0 reduces the power by half.
-executable	`string`	The executable to use (default is vovtasker). For Accelerator, the default is vovtaskerroot.
-expiredate	`string`	Specifies the date and time after which the definition of this tasker is expired, and it cannot be started with vovtaskermgr command. The format of this parameter is year_month_day_hour_min_sec. Example: 2018_12_31_23_59_00
-failover		Passing this option will set the tasker's capacity to 0, which prevents the tasker from accepting jobs and pulling a license. This also acts as a flag to perform some failover configuration testing, such as checking `servercandidates.tcl` to make sure the tasker host is in the list, triggering a check to make sure the host has at least as many file descriptors as vovserver so it can operate at full capacity in the event of failover, and checking that the `server_election` directory is empty.
-host	`hostname`	The name to be used to connect to the vovserver (e.g. "localhost")

Option	Argument	Description
-indirect	`file`	Execute the jobs indirectly, using the Tasker* procedures described in the given file. This is used in indirect taskers.
-maxcapacity	`int`	The capacity of taskers can increase dynamically as a side effect of having suspended jobs. This limits the maximum capacity. The default value is twice the capacity.
-maxjobs	`int`	The maximum number of jobs a tasker can execute. Taskers will become suspended when the max is reached, and will exit once the last job is finished. Default: 0 (unlimited).
-maxidle	`timespec`	The maximum amount of time a tasker can be idle (having no jobs). Default: - (unlimited).
-maxlife	`timespec`	The maximum amount of time a tasker is allowed to run. Default: - (unlimited).
-maxload	`double`	The maximum load for the vovtasker. Above this, its power becomes zero, and the vovtasker does not accept new jobs until the load declines below this value. This helps avoid overloaded machines.
-message	`string`	Message to set on the vovtasker at startup. Should be brief.
-mindisk	`number`	Minimum disk space, in MB (for example, 100) or in percentage (0%-99%, for example, 10%) , on `/usr/tmp` below which the vovtasker will automatically be suspended.
-name	`string`	Name of the vovtasker. The default is the leaf name of the machine on which the vovtasker runs. May not contain the '.' (dot) character.
-nice	`int`	Run the tasker with niceness (reduced OS priority), for UNIX vovtasker only, ignored on Windows.
-power	`double`	The raw power to be used for this vovtasker. The default for this is 0.0, which implies that the raw power is computed automatically upon startup. You can use this to make machines know to be identically- provisioned to have the same power.

Option	Argument	Description
-repeat	`int`	Number of identical vovtasker on a host (obsolete).
-reserve	`reserve expression`	Reserve the vovtasker upon startup. The argument is a reservation expression. Example 1: "`-reserve /john/3w`" This means create a reservation for user john with a duration of three weeks at vovtasker startup. Example 2: "`-reserve memregr//4h`" This means create a reservation for group 'memregr' with a duration of 4h at vovtasker startup.
-resources	`string`	The tasker resources. This could be a list of literals like "bighost maingroup" or contain symbolic values like "@RAM@ @CPUS@". You must restart vovtaskers after changing values specified here. For a method that does not require restart, read about The taskerClass.table file.
-rshcmd	`string`	The command used to start a remote shell on the tasker machine. This is rsh by default, but it could be set, for example, to ssh. The known values for this option are: • rsh, the default value • ssh, the typical value • `vovtsd`
-serverdir	`dir`	Explicit path to the server directory for the tasker.
-taskergroup	`group`	Define the taskergroup in `taskers.tcl`. Often this is used to group similarly-provisioned machines.
-update	`timespec`	The update cycle time (heart beat) of the vovtasker. The default value is 60s. You can use shorter values to cause resources to be updated more frequently when using resource procedures, being mindful of the CPU load this brings to the vovserver.
-vovdir	`dir`	Explicit path to the VOV installation for the tasker.
-vovtsdport	`port`	Port number to be used when connecting to the VOV tasker service daemon, `vovtsd`.

vovtaskermgr

The main way to start, configure, and stop the taskers is with the `vovtaskermgr` command. This command acts relative to the VOV-project enabled in the shell where it is issued.

The file `taskers.tcl` in the `project.swd` directory stores the configuration information used by this command.

Note: Changes made to `taskers.tcl` are not automatically propagated to the running vovtaskers. To do this, use the `update` subcommand.

A vovtasker listed in the `taskers.tcl` file may be running or stopped. The `show` subcommand gives information on the running vovtaskers currently connected to the vovserver. The `list` subcommand gives the names of all the vovtaskers defined in vovtaskers, whether running or stopped.

```
vovtaskermgr: Usage Message

USAGE:
    vovtaskermgr <SUBCOMMAND> [options] [taskerList]

    SUBCOMMAND is case-insensitive.

    The taskerList consists of tasker names or tasker ids.

SUBCOMMAND is one of:
    LIST           -- List all hosts named in the taskers.tcl file.
    RESTART        -- Same as STOP followed by START.
    REFRESH        -- Refresh cached environments and equivalences.
                      The default behavior is for taskers to obtain the
                      equivalences from the server. If changes are made to the
                      equiv.tcl file, the server will need to be instructed to
                      reread the file using the "vovproject reread" command
                      prior to requesting a tasker refresh.
                      If VOVEQUIV_CACHE_FILE is set to "legacy", a host-based
                      equivalence cache file will be created and updated in
                      the SWD/equiv.caches directory. If VOVEQUIV_CACHE_FILE
                      is set to a file path, the specified file will be used
                      instead.
    SHOW           -- Show info about connected or down taskers.
    PRINTSTATUS    -- Tell taskers to print their status in their log file.
    START          -- Start configured taskers.  If a list of hosts is
                      given, start taskers only on those hosts.  Otherwise,
                      start all configured taskers that are not running.
    UPDATE         -- Update configuration of running taskers.
    RESERVE        -- Reserve specified taskers.
    RESERVESHOW    -- Show current tasker reservations.
    CONFIGURE      -- Reconfigure the specified taskers on-the-fly.
                      Changes only persist until the tasker is stopped.
    STOP           -- Stop taskers; let jobs finish, unless -force is given.
    CANCELSHUTDOWN -- Revert stopped but still running taskers to normal
                      so they continue running and accept new jobs.
    ROTATELOG      -- Recreate new log files for specified taskers
                      if log files are missing, create tasker log directories
                      if needed, and have no impact on tasker startup logs.
    CLOSE [MSG]    -- Close taskers from accepting jobs. Closed taskers will
                      start and run, but will do so in a suspended state,
                      displaying the closure message, until opened by the
```

```
                       administrator. The default closure message is
                       'Closed by administrator'.
     OPEN [MSG]     -- Open taskers to accept jobs. The accompanying message
                       will be displayed on running taskers until another
                       message is generated during the course of normal
                       operation. Taskers that are not running will not display
                       the message after starting. The default opening message
                       is an empty string.
Global Options are:
     -l             -- Use longer format with LIST (may be repeated).
     -v             -- Increase verbosity of messages.
     -cfgfile       -- Specify path to tasker config file, relative to SWD.
                       Default: taskers.tcl
     -failover      -- Restrict operation to dedicated failover taskers only.

Options for SHOW are:
     -nameonly      -- Show only the names of the connected taskers.
     -nameid        -- Show only the names and ids of the connected taskers.
     -resourceonly -- Show only the resources of the connected taskers.
     -down          -- Show names of configured taskers that are down.
     -license       -- Show licensed capabilities of connected taskers.
     -taskergroups -- Show tasker group for each connected tasker.

Options for START and RESTART are:
     -server       -- Start the taskers by rsh/ssh from the vovserver host.
                      By default, the taskers are started
                      by the host that executes this script.
     -random       -- Start taskers in random order.
                      This is useful to start a large pool of tasker,
                      by running multiple concurrent commands like:
                        % vovtaskermgr start -random &
                        % vovtaskermgr start -random &
                        % vovtaskermgr start -random &
     -nolog        -- Redirect tasker output to /dev/null.
                      Useful to avoid huge log files in /usr/tmp
     -confirmafter <TIMESPEC>
                   -- Wait for the given time specification after the last start
                      request for the list of taskers being started, then print
                      whether each tasker has successfully started and connected
                      to the vovserver. Only taskers in the READY, WRKNG, FULL, or
                      OVRLD state will be considered as running.

Options for RESERVE are:
     -user         -- Reserve the tasker(s) for given list of users
                       (comma separated list)
     -usergroup    -- Reserve the tasker(s) for given list of user groups
                       (comma separated list)
     -group        -- Reserve the tasker(s) for given list of FairShare groups
                       (comma separated list)
     -jobclass     -- Reserve the tasker(s) for given list of jobclasses
                       (comma separated list)
     -jobproj      -- Reserve the tasker(s) for given list of job projects
                       (comma separated list)
     -osgroup      -- Reserve the tasker(s) for given list of Unix groups
                       (comma separated list)
     -bucketid     -- Reserve the tasker(s) for given list of queue buckets
                       (comma separated list)
     -id           -- Reserve the tasker(s) for given list of jobs
                       (comma separated list of job ids)
     -start        -- Reservation start time
     -end          -- Reservation end time
     -duration     -- Reservation duration (VOV timespec)
     -cancel       -- Cancel the reservation on tasker(s)
```

```
    -hardfill    -- Backfill the reservation with only
                    autokill jobs
    -softfill    -- Backfill the reservation with only
                    autokill or xdur jobs

Options for STOP are:
    -force       -- Stop taskers with force. BEWARE: kills running jobs.
    -noconfirm   -- Do not prompt for confirmation. Default is to prompt.
    -all         -- Stop all running taskers.
    -sick <TIMESPEC>
                 -- Stop all taskers that have been sick for at least the given
                    time specification, as compared against the last time a
                    heartbeat was received by the server for each sick tasker.
                    All jobs running on a sick tasker being stopped will be
                    marked as failed in the server, even if the job does,
                    or has, completed successfully while the tasker is sick.
                    It is recommended to check tasker host connectivity before
                    using this function and allow for the tasker to reconnect
                    and send a heartbeat in case connectivity is restored.

Parameters for CONFIGURE are:
    -allowcoredump <bool>     -- Control core-dump behavior.
    -autokillmethod <d|n|v>   -- Control autokill method.
    -capacity <CAP>[MAXCAP]   -- Specify capacity and optionally the
                                 max-capacity of the tasker. The capacity is
                                 the maximum number of jobs that can be run by
                                 tasker. The max_capacity is the maximum slots
                                 a tasker can be expanded to have when jobs are
                                 suspended. The default value for capacity is
                                 equal to the number of CORES present. The
                                 default value for max_capacity is 2*CAPACITY.
                                 Use N, N/N, CORES[-+*/]N, CORES[-+*/]N/N,
                                 N/CORES[-+*/]N, CORES[-+*/]N/CORES[-+*/]N to
                                 make adjustments from the default.
                                 Examples: 4, 4/8, CORES-2, CORES*0.8,
                                           CORES+0/20, CORES+2/CORES*2
    -cpus <N>                 -- Number of CPUs in this machine.
    -debugcontainers <bool>   -- Enable debug logging of container activity.
    -debugjobcontrol <bool>   -- Enable debug logging of job control activity.
    -debugmultienv   <bool>   -- Enable debug logging of environment switching.
    -debugnuma       <bool>   -- Enable debug logging of NUMA activity.
    -debugusageinfo  <bool>   -- Enable debug logging of memory usage analysis.
    -maxload <MAXLOAD>        -- Maximum load above which new jobs are refused.
                                 The default value for max_load is
                                 CAPACITY+0.5.
                                 Use 0 or less than 0 to specify default value.
                                 Use N or CAPACITY[-+*/]N to make adjustments
                                 from the default.
                                 Examples: 12.0, CAPACITY+2, CAPACITY*2
    -maxwaitnostart <N>       -- How long to wait for a job to start.
    -maxwaittoreconnect <N>   -- How long to wait before reconnect.
    -message <string>         -- Set vovtasker message.
    -numabindtonode <bool>    -- Bind to entire NUMA node or individual cores.
                                 Default is to bind to entire NUMA node.
    -resources  <string>      -- vovtasker resources.
    -taskergroup <string>     -- The tasker group.
    -minramfree <N>           -- Minimum amount of free RAM in MB.
    -name <string>            -- Name of vovtasker.
    -ramsentry <bool>         -- Activate/Deactivate RAM SENTRY.
    -efftotram <N>            -- Effective total RAM in MB.
    -retrychdir <N>           -- Specify number of retries for failed chdirs.
    -retrychdirsleep <N>      -- Specify the sleep interval time between
                                 retries for failed chdirs.
    -retrychdirbackoff <N>   -- Specify the factor multiplied to the sleep
                                interval to increase sleep interval between
                                retries for failed chdirs.
    -liverecorder on|off     -- Enable/disable LiveRecorder debugging
                                capability (linux64 only).
    -liverecorder.logdir <string>
                             -- Specify the directory in which the LiveRecorder
                                recording file should be saved. The
                                directory must exist. Default is "/tmp".
    -liverecorder.logsize <N> --
                                Specify the LiveRecorder log size in MB.
                                Default: 256, Min: 256, Max: 65536.
    -liverecorder.mode <string>
                             -- Specify the LiveRecorder mode, which is one of
                                the following: tasker, subtasker, both.
                                Note that enabling subtasker recording results
                                in a recording file for each job executed on
                                the tasker.
                                Default: tasker.
    -rawpower                -- Specify a raw power figure for initial tasker
                                startup.
    -mindisk                 -- Specify minimum /tmp disk in MB or
                                percentage (0%-99%, for example, 10%)
                                required for tasker startup.
    -coeff                   -- Specify a scaling factor from 0.01-100.0
                                used to derate tasker power.
    -sendenv <name>          -- Send a named environment to a tasker.
    -setenv VAR=VALU  E      -- Set a variable in the tasker environment.
                                ("VAR=VALUE" must be quoted on Windows)
    -taskerheartbeat <N>     -- Specify the heartbeat for a tasker.
    -unsetenv VAR            -- Unset a variable in the tasker environment.
    -hardbound <bool>        -- Dispatch autokill jobs only.
    -softbound <bool>        -- Dispatch autokill or xdur jobs only.

EXAMPLES:
    % vovtaskermgr show
    % vovtaskermgr show -nameid
    % vovtaskermgr start
    % vovtaskermgr start unix1
    % vovtaskermgr start -random            -- Start taskers in random order.
    % vovtaskermgr update
    % vovtaskermgr restart
    % vovtaskermgr stop                     -- Stop all taskers, let running
                                               jobs finish.
    % vovtaskermgr stop -noconfirm          -- Like above, no confirmation
                                               required.
    % vovtaskermgr stop -force              -- Kill running jobs now
                                               (-noconfirm implied).
    % vovtaskermgr reserve -user john \\
             -duration 3h jupiter           -- Reserve tasker jupiter for user
                                               john for 3h from now
    % vovtaskermgr configure -message "shutdown 1PM" farm11 farm12
    % vovtaskermgr printstatus farm11
    % vovtaskermgr rotatelog                -- Recreate missing log files for
                                               all connected taskers
    % vovtaskermgr rotatelog farm2 farm11   -- Recreate missing log files for
                                               tasker farm2 farm11
    % vovtaskermgr configure jupiter -sendenv BASE
                                            -- send the BASE environment to
                                               tasker jupiter
    % vovtaskermgr reserve -hardfill -user john \\
             -duration 3h jupiter           -- Reserve tasker jupiter for user
```

```
                                                  john for 3h from now, and backfill
with autokill jobs
   % vovtaskermgr configure jupiter -softbound 1
                                               -- Reserve tasker jupiter for jobs with
autokill or xdur specified
```

Starting Many Taskers in Parallel

If you have hundreds of taskers to start, it may take some time. You can speed up the process by running multiple start script with the -random option, which is useful to start taskers in random order.

For example:

```
% vovtaskermgr start -random &
% vovtaskermgr start -random &
% vovtaskermgr start -random &
% vovtaskermgr start -random &
% vovtaskermgr start -random &
% vovtaskermgr start -random &
```

Tasker Configuration on the Fly

Many vovtasker characteristics can be changed on the fly using `vovtaskermgr configure`. For example, you can change the capacity of a tasker, i.e. the maximum number of jobs that the tasker can take, with:

```
% vovtaskermgr configure -capacity 8 pluto
```

Setting the capacity to zero effectively disables the tasker:

```
% vovtaskermgr configure -capacity 0 pluto
% vovtaskermgr configure -message "Temporarily disabled by John" pluto
```

Tasker Capacity

The behavior of manually overriding vovtasker cores and capacity has been improved. By default, the capacity follows the core count, but it can also be manually set via the -T option or by defining the SLOTS/N consumable resource via the -r option, where N is a positive integer. In all cases, the capacity directly affects the number of slot licenses that will be requested.

Tasker Reservation

Below is an example of using `vovtaskermgr` to set a reservation on a tasker. In this case, you want to reserve the tasker called 'pluto' for user 'john' for 2 days.

If you wish for the vovtaskers to be reserved when they start, use the -reserve option in the `taskers.tcl` file.

```
% vovtaskermgr reserve -user john -duration 2d pluto
```

vovtaskermgr

```
vovtaskermgr: Usage Message

USAGE:
    vovtaskermgr <SUBCOMMAND> [options] [taskerList]

    SUBCOMMAND is case-insensitive.

    The taskerList consists of tasker names or tasker ids.

SUBCOMMAND is one of:
    LIST            -- List all hosts named in the taskers.tcl file.
    RESTART         -- Same as STOP followed by START.
    REFRESH         -- Refresh cached environments and equivalences.
                       The default behavior is for taskers to obtain the
                       equivalences from the server. If changes are made to the
                       equiv.tcl file, the server will need to be instructed to
                       reread the file using the "vovproject reread" command
                       prior to requesting a tasker refresh.
                       If VOVEQUIV_CACHE_FILE is set to "legacy", a host-based
                       equivalence cache file will be created and updated in
                       the SWD/equiv.caches directory. If VOVEQUIV_CACHE_FILE
                       is set to a file path, the specified file will be used
                       instead.
    SHOW            -- Show info about connected or down taskers.
    PRINTSTATUS     -- Tell taskers to print their status in their log file.
    START           -- Start configured taskers.  If a list of hosts is
                       given, start taskers only on those hosts.  Otherwise,
                       start all configured taskers that are not running.
    UPDATE          -- Update configuration of running taskers.
    RESERVE         -- Reserve specified taskers.
    RESERVESHOW     -- Show current tasker reservations.
    CONFIGURE       -- Reconfigure the specified taskers on-the-fly.
                       Changes only persist until the tasker is stopped.
    STOP            -- Stop taskers; let jobs finish, unless -force is given.
    CANCELSHUTDOWN -- Revert stopped but still running taskers to normal
                       so they continue running and accept new jobs.
    ROTATELOG       -- Recreate new log files for specified taskers
                       if log files are missing, create tasker log directories
                       if needed, and have no impact on tasker startup logs.
    CLOSE [MSG]     -- Close taskers from accepting jobs. Closed taskers will
                       start and run, but will do so in a suspended state,
                       displaying the closure message, until opened by the
                       administrator. The default closure message is
                       'Closed by administrator'.
    OPEN [MSG]      -- Open taskers to accept jobs. The accompanying message
                       will be displayed on running taskers until another
                       message is generated during the course of normal
                       operation. Taskers that are not running will not display
                       the message after starting. The default opening message
                       is an empty string.
Global Options are:
    -l             -- Use longer format with LIST (may be repeated).
    -v             -- Increase verbosity of messages.
    -cfgfile       -- Specify path to tasker config file, relative to SWD.
                      Default: taskers.tcl
    -failover      -- Restrict operation to dedicated failover taskers only.
```

```
Options for SHOW are:
    -nameonly     -- Show only the names of the connected taskers.
    -nameid       -- Show only the names and ids of the connected taskers.
    -resourceonly -- Show only the resources of the connected taskers.
    -down         -- Show names of configured taskers that are down.
    -license      -- Show licensed capabilities of connected taskers.
    -taskergroups -- Show tasker group for each connected tasker.

Options for START and RESTART are:
    -server      -- Start the taskers by rsh/ssh from the vovserver host.
                    By default, the taskers are started
                    by the host that executes this script.
    -random      -- Start taskers in random order.
                    This is useful to start a large pool of tasker,
                    by running multiple concurrent commands like:
                      % vovtaskermgr start -random &
                      % vovtaskermgr start -random &
                      % vovtaskermgr start -random &
    -nolog       -- Redirect tasker output to /dev/null.
                    Useful to avoid huge log files in /usr/tmp
    -confirmafter <TIMESPEC>
                 -- Wait for the given time specification after the last start
                    request for the list of taskers being started, then print
                    whether each tasker has successfully started and connected
                    to the vovserver. Only taskers in the READY, WRKNG, FULL, or
                    OVRLD state will be considered as running.

Options for RESERVE are:
    -user        -- Reserve the tasker(s) for given list of users
                    (comma separated list)
    -usergroup   -- Reserve the tasker(s) for given list of user groups
                    (comma separated list)
    -group       -- Reserve the tasker(s) for given list of FairShare groups
                    (comma separated list)
    -jobclass    -- Reserve the tasker(s) for given list of jobclasses
                    (comma separated list)
    -jobproj     -- Reserve the tasker(s) for given list of job projects
                    (comma separated list)
    -osgroup     -- Reserve the tasker(s) for given list of Unix groups
                    (comma separated list)
    -bucketid    -- Reserve the tasker(s) for given list of queue buckets
                    (comma separated list)
    -id          -- Reserve the tasker(s) for given list of jobs
                    (comma separated list of job ids)
    -start       -- Reservation start time
    -end         -- Reservation end time
    -duration    -- Reservation duration (VOV timespec)
    -cancel      -- Cancel the reservation on tasker(s)
    -hardfill    -- Backfill the reservation with only
                    autokill jobs
    -softfill    -- Backfill the reservation with only
                    autokill or xdur jobs

Options for STOP are:
    -force       -- Stop taskers with force. BEWARE: kills running jobs.
    -noconfirm   -- Do not prompt for confirmation. Default is to prompt.
    -all         -- Stop all running taskers.
    -sick <TIMESPEC>
                 -- Stop all taskers that have been sick for at least the given
                    time specification, as compared against the last time a
                    heartbeat was received by the server for each sick tasker.
                    All jobs running on a sick tasker being stopped will be
                    marked as failed in the server, even if the job does,
                    or has, completed successfully while the tasker is sick.
                    It is recommended to check tasker host connectivity before
                    using this function and allow for the tasker to reconnect
                    and send a heartbeat in case connectivity is restored.

Parameters for CONFIGURE are:
    -allowcoredump <bool>     -- Control core-dump behavior.
    -autokillmethod <d|n|v>   -- Control autokill method.
    -capacity <CAP>[MAXCAP]   -- Specify capacity and optionally the
                                 max-capacity of the tasker. The capacity is
                                 the maximum number of jobs that can be run by
                                 tasker. The max_capacity is the maximum slots
                                 a tasker can be expanded to have when jobs are
                                 suspended. The default value for capacity is
                                 equal to the number of CORES present. The
                                 default value for max_capacity is 2*CAPACITY.
                                 Use N, N/N, CORES[-+*/]N, CORES[-+*/]N/N,
                                 N/CORES[-+*/]N, CORES[-+*/]N/CORES[-+*/]N to
                                 make adjustments from the default.
                                 Examples: 4, 4/8, CORES-2, CORES*0.8,
                                           CORES+0/20, CORES+2/CORES*2
    -cpus <N>                 -- Number of CPUs in this machine.
    -debugcontainers <bool>   -- Enable debug logging of container activity.
    -debugjobcontrol <bool>   -- Enable debug logging of job control activity.
    -debugmultienv   <bool>   -- Enable debug logging of environment switching.
    -debugnuma       <bool>   -- Enable debug logging of NUMA activity.
    -debugusageinfo  <bool>   -- Enable debug logging of memory usage analysis.
    -maxload <MAXLOAD>        -- Maximum load above which new jobs are refused.
                                 The default value for max_load is
                                 CAPACITY+0.5.
                                 Use 0 or less than 0 to specify default value.
                                 Use N or CAPACITY[-+*/]N to make adjustments
                                 from the default.
                                 Examples: 12.0, CAPACITY+2, CAPACITY*2
    -maxwaitnostart <N>       -- How long to wait for a job to start.
    -maxwaittoreconnect <N>   -- How long to wait before reconnect.
    -message <string>         -- Set vovtasker message.
    -numabindtonode <bool>    -- Bind to entire NUMA node or individual cores.
                                 Default is to bind to entire NUMA node.
    -resources  <string>      -- vovtasker resources.
    -taskergroup <string>     -- The tasker group.
    -minramfree <N>           -- Minimum amount of free RAM in MB.
    -name <string>            -- Name of vovtasker.
    -ramsentry <bool>         -- Activate/Deactivate RAM SENTRY.
    -efftotram <N>            -- Effective total RAM in MB.
    -retrychdir <N>           -- Specify number of retries for failed chdirs.
    -retrychdirsleep <N>      -- Specify the sleep interval time between
                                 retries for failed chdirs.
    -retrychdirbackoff <N>    -- Specify the factor multiplied to the sleep
                                 interval to increase sleep interval between
                                 retries for failed chdirs.
    -liverecorder on|off      -- Enable/disable LiveRecorder debugging
                                 capability (linux64 only).
    -liverecorder.logdir <string>
                              -- Specify the directory in which the LiveRecorder
                                 recording file should be saved. The
                                 directory must exist. Default is "/tmp".
    -liverecorder.logsize <N> --
                                 Specify the LiveRecorder log size in MB.
                                 Default: 256, Min: 256, Max: 65536.
    -liverecorder.mode <string>
                              -- Specify the LiveRecorder mode, which is one of
                                 the following: tasker, subtasker, both.
```

```
                                Note that enabling subtasker recording results
                                in a recording file for each job executed on
                                the tasker.
                                Default: tasker.
    -rawpower                -- Specify a raw power figure for initial tasker
                                startup.
    -mindisk                 -- Specify minimum /tmp disk in MB or
                                percentage (0%-99%, for example, 10%)
                                required for tasker startup.
    -coeff                   -- Specify a scaling factor from 0.01-100.0
                                used to derate tasker power.
    -sendenv <name>          -- Send a named environment to a tasker.
    -setenv VAR=VALU  E      -- Set a variable in the tasker environment.
                                ("VAR=VALUE" must be quoted on Windows)
    -taskerheartbeat <N>     -- Specify the heartbeat for a tasker.
    -unsetenv VAR            -- Unset a variable in the tasker environment.
    -hardbound <bool>        -- Dispatch autokill jobs only.
    -softbound <bool>        -- Dispatch autokill or xdur jobs only.

EXAMPLES:
    % vovtaskermgr show
    % vovtaskermgr show -nameid
    % vovtaskermgr start
    % vovtaskermgr start unix1
    % vovtaskermgr start -random             -- Start taskers in random order.
    % vovtaskermgr update
    % vovtaskermgr restart
    % vovtaskermgr stop                      -- Stop all taskers, let running
                                                jobs finish.
    % vovtaskermgr stop -noconfirm           -- Like above, no confirmation
                                                required.
    % vovtaskermgr stop -force               -- Kill running jobs now
                                                (-noconfirm implied).
    % vovtaskermgr reserve -user john \\
             -duration 3h jupiter            -- Reserve tasker jupiter for user
                                                john for 3h from now
    % vovtaskermgr configure -message "shutdown 1PM" farm11 farm12
    % vovtaskermgr printstatus farm11
    % vovtaskermgr rotatelog                 -- Recreate missing log files for
                                                all connected taskers
    % vovtaskermgr rotatelog farm2 farm11    -- Recreate missing log files for
                                                tasker farm2 farm11
    % vovtaskermgr configure jupiter -sendenv BASE
                                             -- send the BASE environment to
                                                tasker jupiter
    % vovtaskermgr reserve -hardfill -user john \\
             -duration 3h jupiter            -- Reserve tasker jupiter for user
                                                john for 3h from now, and backfill
 with autokill jobs
    % vovtaskermgr configure jupiter -softbound 1
                                             -- Reserve tasker jupiter for jobs with
 autokill or xdur specified
```

Tasker ID

Each tasker is an object in the VOV database and therefore it has its own VovId.

This ID is used in the following commands:

- vovtaskermgr (for command line operation)
- vtk_job_control to stop, suspend, resume jobs
- vtk_tasker_reserve to reserve a tasker.

Tasker Attributes

A tasker is characterized by several attributes. These attributes are controllable by means of the command line arguments to the vovtasker binary as well as by means of the procedure vtk_tasker_define in the `taskers.tcl` configuration file for a VOV project.

For cases where the vovtaskers are started by submitting the binary to a separate batch queue system, the system manager may create a copy of the vovtasker binary called `vovagent`. When invoked by this name, the binary will limit the values of some attributes based on information stored in the configuration file `$VOVDIR/local/vovagent.cfg`.

Attribute Name	vovtasker options	vtk_tasker_define option	Description
name	-a	-name	Name of the tasker. By default, it is the name of the host on which the tasker is running. The tasker name can contain alphanumeric characters, dashes and underscores. It must be less than 50 characters long. If the name ends with a "_r", it probably indicates a tasker that has reconnected to the tasker after a server crash. These taskers are used to terminate the jobs executing at the time of the crash. The name of a running tasker can be changed with `vovtaskermgr configure -name ...` or using the API `vtk_tasker_config $TASKERID name "newname"`.
capability	-b	-capabilities	The capabilities that the tasker has. This results in license checkout

Attribute Name	vovtasker options	vtk_tasker_define option	Description
			attempts for the specified capabilities. Possible values: FULL, NC, PROCINFO, NETINFO, EXEC, RT. FULL includes all capabilities. Accelerator contains PROCINFO and NETINFO capabilities. Default is FULL.
capacity	-T	-capacity	Maximum number of jobs that can be run by the tasker concurrently. Default is 1 slot per core detected or specified (see -C below).
maxload	-M	-maxload	Maximum allowed load on the tasker host (default N+0.5, where *N* is the number of cores detected). The maximum load is the point at which the host of the tasker is too busy to accept any more jobs. A machine is considered overloaded if its load average for either the last minute or the last five minutes exceeds this boundary.
loadsensor	-L	-loadsensor	Use an SGE style load sensors to control power of a tasker and other resources.
coefficient	-c	-coeff	The tasker coefficient (positive floating point number, with default 1.0). This attribute is used to adjust the raw power. A coefficient of 1.0 indicates the actual computed power of the tasker should be used. A coefficient of 4.0 indicates the actual power of the tasker should be divided by 4.
cpus	-C	-cpus	The number of CPUs in this machine. Default is the number of cores reported by the operating system. This attribute affects the computation of the actual load on the machine and by default, defines the capacity of the tasker (see -T above).
logfile	-l	-logfile	The name of the file for the tasker log. The file name is relative to the server working directory. The file name can contain also the following

Attribute Name	**vovtasker options**	**vtk_tasker_define option**	**Description**
			symbolic strings, which will be appropriately substituted: @NUMBER@ @TASKERHOST@ @SERVERHOST@ @PROJECT@.
reserve	-e	-reserve	The reservation expression for this tasker. The argument is in the format "GROUP/USER/DURATION", where the GROUP and USER fields are optional. Examples: `/john/2wusers//100dregression/john/2w`
resources	-r	-resources	The tasker resources offered by this tasker. The resource management determines the type of jobs the tasker may accept.
Remote Shell Command	n/a	-rshcmd	The command used to start a remote shell tasker on the tasker machine. The default is `rsh` (or `remsh`). Other possible values are: • `ssh` (most common value) • `vovtsd` (useful for Windows taskers), which requires `vovtsd` to be running on the remote machine.
Tasker Environment	n/a	-taskerenv	A space-separated list of VAR=VALUE elements that specify additional environment variables that need to be defined when starting the vovtasker. This parameter is only active when the tasker is started. To change the environment in a running vovtasker you have to use the vtk_tasker_config API.
Tasker Group	n/a	-taskergroup	The group(s) the tasker is in for purpose of viewing in the browser UI or the vovmonitor. This currently has no use other than making it easier to view groups of taskers.
timeleft	N/A	N/A	The time the tasker has left before it suspends itself. Also, the maximum expected duration for a job dispatched

Attribute Name	vovtasker options	vtk_tasker_define option	Description
			to this tasker. This attribute is available in the context of time variant resources and is controlled only by means of the procedure vtk_tasker_set_timeleft.
transient	-i	-transient	A transient tasker is destroyed when the vovtasker client is terminated. On the other hand, a non-transient tasker persists in memory even when vovtasker is terminated. Its status will be "DOWN".
Use Vovfire	-E	-usevovfire	With this option, a direct tasker may use `vovfire` to execute jobs instead of the direct execution of the job. Since `vovfire` does the directory change and the setting of the environment, the tasker does less work. The environment caching of the tasker is disabled in this mode. This is a development option that is useful mostly on Windows.
update	-U	-update	The period used by the tasker to update its status. The argument is in seconds. The default is 60 seconds.
mindisk	-D	-mindisk	The amount of free disk on `/usr/tmp` below which the vovtasker is suspended. The default is 5MB. Many system commands and some VOV ones depend on scratch space here. **Note:** The value can be set to 0 to turn off tasker suspension, but incorrect operation of some commands may occur.
maxidle	-z	N/A	After being idle for the given time, tasker does not accept new jobs and exits after completing active jobs. Value is a FlowTracer timespec, for example, 2m. If unspecified, idle time is unlimited.

Attribute Name	**vovtasker options**	**vtk_tasker_define option**	**Description**
maxlife	-Z	N/A	After the specified lifetime, tasker does not accept new jobs and exits after completing active jobs. Value is a FlowTracer timespec, for example, 2H. If unspecified, lifetime is unlimited.
maxjobs	-m	-maxjobs	The maximum number of jobs a vovtasker will execute during its lifetime. When the max is reached, vovtasker self-suspends so it stops accepting new jobs, and will exit once its last running job finishes. Default: 0 (unlimited).

Tasker Configuration Parameters

The following table shows the vovtasker parameters that can be configured. All parameters can be configured with `vtk_tasker_config` and most parameters can be configured with `vovtaskermgr configure ...`

Parameter Name	**Values**	**Description**
allowcoredump	boolean	Allow dumping of core on error (normally off)
capacity	non-negative	Change number of slots in vovtasker
coeff	0.01 -- 100.0	The tasker coefficient (used to derate power)
debugnuma	boolean	Print debugging messages about NUMA control
efftotram		Effective Total RAM
expirelogs	timespec	Force logs of tasker to expire after specified duration. Default expiration is next midnight

Parameter Name	Values	Description
maxload	positive double	Change max load allowed on tasker
maxwaitnostart	timespec	Time to wait before killing a job that does not want to start. Default is 1m.
message,sys	message	
message,usr	message	
message	message	
mindisk	integer	Minimum disk required in /tmp (in MB) to accept a job
minramfree		Minimum RAM that has to be free to accept a job
name	New tasker name	Change name of tasker
printstatus	1	Print status of tasker
ramsentry	boolean	Activate RAM Sentry mechanism
rawpower	int	Change raw power of tasker
refresh	(ignored)	Force tasker to refresh environment and clear all caches
resources		Change tasker resources
shutdowncancel	(ignored)	Cancel a shutdown order.
taskergroup	name of tasker group	Change group of the tasker
update		Change update time for tasker (default 1m)
verbose	0,1,2,3,4	Change verbosity level (for debugging)
waitafterjc	0-30	Time to wait after a job control action
waitafterjcext	0-30	Time to wait after a EXT job control action

Tasker Power

Upon startup, the vovtasker computes the *raw power* of the CPU on the current host by timing a known test routine consisting of a balanced mix of integer, double and string operations. The power is inversely proportional to the time it takes to execute the test routine.

To maintain the efficiency of the vovserver for other tasks (such as scheduling), only the 2 most significant digits of the calculations are used to search for and identify the fastest tasker host.

Here is a sample output of the startup phase of a tasker, which shows the results of the power measurements:

```
% vovtasker -N
vovtasker Jan 27 17:49:03
        Copyright © 1995-2025, Altair Engineering
        Sun5/5.2 Jan  5 2000 08:28:50
        evcdrom@mercury
vovtasker Date Stamp: Thu Jan 27 17:49:04 2000
vovtasker Jan 27 17:49:04 Test 1: INTEGER OPS  W=  1.00 Reps= 500 T=  16.95ms
vovtasker Jan 27 17:49:04 Test 1: DOUBLE  OPS  W=  1.00 Reps=  25 T=  73.64ms
vovtasker Jan 27 17:49:04 Test 1: CHAR    OPS  W=  0.10 Reps=  10 T=  74.12ms
vovtasker Jan 27 17:49:04 ---- Weighted time: 98.01ms
vovtasker Jan 27 17:49:04 Test 2: INTEGER OPS  W=  1.00 Reps= 500 T=  17.45ms
vovtasker Jan 27 17:49:04 Test 2: DOUBLE  OPS  W=  1.00 Reps=  25 T=  73.36ms
vovtasker Jan 27 17:49:04 Test 2: CHAR    OPS  W=  0.10 Reps=  10 T=  74.68ms
vovtasker Jan 27 17:49:04 ---- Weighted time: 98.27ms
vovtasker Jan 27 17:49:04 Test 3: INTEGER OPS  W=  1.00 Reps= 500 T=  17.52ms
vovtasker Jan 27 17:49:04 Test 3: DOUBLE  OPS  W=  1.00 Reps=  25 T=  73.52ms
vovtasker Jan 27 17:49:04 Test 3: CHAR    OPS  W=  0.10 Reps=  10 T=  72.15ms
vovtasker Jan 27 17:49:04 ---- Weighted time: 98.26ms
vovtasker Jan 27 17:49:04 Best weighted time: 98.01ms
```

This method is very effective in discriminating among CPUs with the same architecture, such as all Sparcs. If you are running on a heterogeneous network, you will notice that the test routine executes very well on PowerPCs and x86 processors compared to Sparc processors. To compensate this effect, we recommended setting the `tasker coefficient` on PowerPCs and x86 hosts to a value greater than 1.0 (e.g. 2.0).

On a multiprocessor machine, "raw power" refers to the power of each single CPU.

The *current power* of a machine takes into account both the raw power and the load (as computed by the UNIX utility `uptime`). On a multiprocessor, if the load is less than the number of CPUs minus one, the current power is the same as the raw power, because there is a processor that is practically idle.

The *effective power* takes into account the "self load" created by the jobs that have been initiated by the tasker itself.

Tasker RAM Saturation

The vovtasker implements the following features related to the exhaustion of memory on a tasker machine:

- RAM saturation occurs when the jobs executing on a tasker take up more memory than available on the machine so that the jobs may start swapping. For this computation use the "Virtual Memory" number and not the "Resident Set Size" number. When the tasker goes into saturation, the tasker orders all jobs by size and suspends enough jobs starting from the small ones to eliminate the saturation condition. The largest job is always left running, regardless of its size.
- If the free RAM on a tasker falls below a threshold, typically below 50MB, the tasker stops accepting jobs.
- The ram sentry mechanism, described in a separate section.

The behavior can currently be controlled only with the API `vtk_tasker_config` by specifying either the parameter *minRamFree* (Minimum Free RAM) or the parameter *effTotRam* (Effective Total RAM), both of which are expressed in MB (1MB = 1,048,576Bytes).

The default for *minRamFree* is 50 or 5% of the total RAM, whichever is less. Example:

```
% vovsh -x "vtk_tasker_config $taskerId  minRamFree  20"
```

The default for *effTotRam* is the total RAM. Example:

```
% vovsh -x "vtk_tasker_config $taskerId  effTotRam  16000"
```

Connect a Single Tasker

You can start a tasker at any time. Try the following commands and check the effect on the tasker monitor.

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] [-K] [-l rootOfDailyLogFile] [-L <loadSensor>]
                     [-m <integer>] [-M max_load] [-n <integer>] [-N] [-o local
                     resources] [-p project] [-P <double>] [-q <hardbound>]
                     [-Q <softbound>] [-r resources] [-R resources] [-s]
                     [-S resources] [-t timeout] [-T capacity[/max_capacity]]
                     [-U <CSV list of timeSpecs>] [-v number]
                     [-V ncName@ncHost[:port]] [-w WX properties] [-W Tasker is
                     a member of a union.] [-x Experimental; do not use.]
                     [-z <timeSpec>] [-Z <timeSpec>]
   -A:          The name of 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 NETINFO EXEC RT
                  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.
    -d:         Activate debugging
    -D:         Min disk space in MB in /tmp and /usr/tmp (default 5)
    -e:         Reserve tasker from the beginning: format of reserveExpr is
                either the old 'GroupName/UserName/Duration' or the new list of
                'keyword value' pairs where the keywords are USER GROUP
                JOBCLASS JOBPROJ BUCKET ID DUR TRANSIENT. -e "JOBCLASS c1 DUR
                1d TRANSIENT 1" will reserve the tasker for JOBCLASS c1 for 1
                day. When the tasker disconnects,the reservation will be
                removed as well
    -E:         Use vovfire to execute jobs: valid for direct taskers only.
                Disables caching of environments
    -f:         Source the given Tcl file
    -F:         ncTasker config file.
    -g:         Set the taskergroup for this tasker
    -G:         Specify Fairshare Group used by an indirect tasker. Use with -V.
    -h:         Host    (default is env. var. VOV_HOST_NAME)
    -H:         Select health checks you want:
                  P / p     Enable/Disable portmap check
                  D / d     Enable/Disable disk space check
                  W / w     Enable/Disable writability for /tmp and /usr/tmp
                  U / u     Enable/Disable user script check
                   ($VOVDIR/local/tasker/health_user_script.sh)
                 Example:  -H pDWu
    -i:         Make the tasker transient (-i 1) or persistent (-i 0). If a
                tasker is transient (the default), it is destroyed when the
                client disconnects. Persistent taskers must be 'indirect'
                taskers (see -I flag).
    -I:         Indirect execution mode. The argument indicates the file that
                describes the procedures to start and stop jobs indirectly. See
                the Reference Manual for more info. If the argument is just a
                dash '-' the option is ignored.
    -j:         Disable job statistics (useful on machines with lots of CPUS)
    -k:         Specify autokill mode (d=direct, n=ncstop, v=vovstop). Default
                'd'
    -K:         Use Quick Connect
    -l:         Specify root of daily log file. The actual logs will be of the
                form FFFF_YYYY.MM.DD.log. Also closes stdin.
    -L:         Specify a SGE-like Load Sensor
    -m:         Maximum number of jobs allowed to run on tasker.
    -M:         Maximum allowed load on the tasker host. The default value for
                max_load is CAPACITY+0.5. Use 0 or less than 0 to specify
                default value. Use N or CAPACITY[-+*/]N to make adjustments
                from the default. Examples: -M 12.0, -M CAPACITY+2, -M
                CAPACITY*2
    -n:         Run in nice mode with lower CPU priority
    -N:         Normal tasker. Same as -r @STD@
    -o:         Local resources (vovwxd internal)
    -p:         Project (default is env. var. VOV_PROJECT_NAME)
    -P:         Specify raw power of tasker, instead of computing it
                automatically.
    -q:         Dispatch only autokill job to hardbound tasker tasker.
    -Q:         Dispatch autokill and xdur job to softbound tasker tasker.
    -r:         Resources for this tasker. This can be either a list of
                resources or a Tcl expression that calls a procedure in the
                VovResources:: namespace. To simplify scripting, it is also
                possible to encode the resource string with base64 and pass the
                encoded string XXXX with -r base64:XXXX (i.e. no need to quote
                spaces in argument to -r option)
    -R:         Resources appended to the jobs by the agent. Use only with -V
    -s:         (OBSOLETE) Silent mode. Also closes stdin.
    -S:         Resource filter on what NC taskers to attach to the agent. Use
                only with -V
    -t:         Try multiple times to connect to server. Give up only after
                'timeout' seconds.
    -T:         Specify capacity and optionally the max-capacity of the tasker.
                The capacity is the maximum number of jobs that can be run by
                tasker. The max_capacity is the maximum slots a tasker can be
                expanded to have when jobs are suspended. The default value for
                capacity is equal to the number of CORES present. The default
                value for max_capacity is 2*CAPACITY. Use -1 or 'auto' to
                specify the default. Use N, N/N, CORES[-+*/]N, CORES[-+*/]N/N,
                N/CORES[-+*/]N, CORES[-+*/]N/CORES[-+*/]N to make adjustments
                from the default. Examples: -T 4, -T 4/8, -T AUTO/1000, -T
                CORES-2, -T CORES*0.8, -T CORES+0/20, -T CORES+2/CORES*2
    -U:         Update intervals for resources, tasker statistics, and job
                statistics. The resources update interval is also known as the
                tasker heartbeat. One (resources), two (resources,taskerStats),
                or three (resources,taskerStats,jobStats) values may be
                specified. Defaults: 60s,120s,30s
    -v:         Set verbose level (0-4): default 1.
    -V:         NetworkComputer indirect tasker
    -w:         Reserved for system use
    -W:         Reserved for system use
    -x:         Experimental; do not use.
    -z:         After being idle for given time, tasker exits.
    -Z:         After specified lifetime, tasker does not accept new jobs and
                exits after completing active jobs.
```

Examples

```
% vovtasker -h                -- Get the usage message.
% vovtasker -N &              -- Start a normal tasker.
% vovtasker -r @STD@ &        -- Same as vovtasker -N.
% vovtasker -r foo &          -- Only resource "foo" is offered.
% vovtasker -r "foo @STD@" &  -- Resource "foo" is offered
                                in addition to all the default resources.
% vovtasker -M 20 &           -- Add a tasker with a huge MAX-LOAD of 20.
```

Tasker Status

The following table applies to "direct" taskers and to "BPS agents".

Status	Color	Description
Idle	green	The tasker is ready to run jobs.
Working	dark yellow	The tasker is running some jobs and it has some additional capacity.

Status	**Color**	**Description**
Full	light yellow	The tasker is running as many jobs as it is capable.
Overloaded	red	The load on the host where the tasker is running is higher than the maximum load allowed by the tasker.
Busy	light gray	The tasker is doing some bookkeeping operation.
Suspended	purple	The tasker has been suspended and it is not accepting any new jobs.
Paused	pale purple	The tasker has been paused by a signal from an external process. It can be awakened by sending a SIGCONT signal.
Sick	black	The tasker has not sent the heartbeat for more than twice its update interval (update interval is 1 minute by default).
Requested (previously called Down)	dark gray	The tasker is persistent (not transient) and the associated client is currently not connected.
Done	blue	The tasker is about to exit and no longer accepting jobs.
NoLic	gray	The tasker is waiting for a license.

Status of BPS Taskers

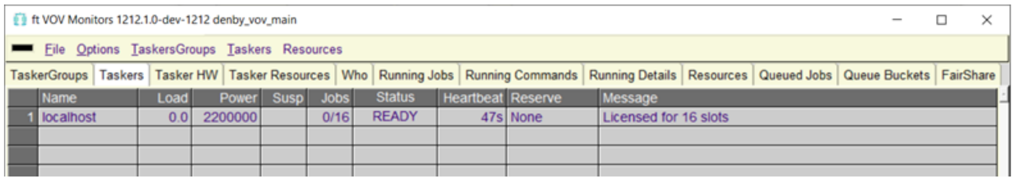
A BPS tasker is a tasker that represent a computing host that belongs to another Batch Processing System (e.g. SGE or Accelerator).

Status	**Color**	**Description**
Ok	green	Not in an error condition.

Status	Color	Description
Warning	red	Used for BPS taskers. The BPS system reports some problems with this tasker.
Unknown	gray	The BPS system has not reported enough information about this host.

Monitor Taskers

To monitor the activity of the taskers, use either `vsm` from the CLI, or click **Console** > **Vov Monitor** from the GUI.


ft VOV Monitors 1212.1.0-dev-1212 denby_vov_main
File Options TaskersGroups Taskers Resources
TaskerGroups Taskers Tasker HW Tasker Resources Who Running Jobs Running Commands Running Details Resources Queued Jobs Queue Buckets FairShare

	Name	Load	Power	Susp	Jobs	Status	Heartbeat	Reserve	Message
1	localhost	0.0	2200000		0/16	READY	47s	None	Licensed for 16 slots

Figure 2:

From the command line, you can use:

```
% vovtaskermgr show
1 04204992   bellevue 0.00    90909 0/1 READY  Unlim.
2 04205717       mars 0.00    39239 0/1 READY  Unlim.
3 04205719     saturn 0.05    10633 0/1 READY  Unlim.
4 04205720       moon 0.02    15003 0/1 READY  Unlim.
5 04205721     cayman 0.00    83333 0/1 READY  Unlim.
6 04205722      comet 0.07    58595 0/1 READY  Unlim.
7 04205725    cheetah 0.05   224089 0/1 READY  Unlim.
8 04205726    jupiter 0.03    13404 0/2 READY  Unlim.
```

From the browser interface, go to the Tasker page. For a large number of taskers, such as hundreds or more, you may find the Tasker page more compact and convenient.

Run vovtasker monitor with CLI vsm

The command `vsm` is used to run the Tasker Monitor in its own window. This can also be called from the VovConsole or from any command line that has the right environment and is enabled for the project.

Tasker Resources

The resources offered by a tasker are represented by a space-separated list of tokens. Resources can be **explicit** or **symbolic**.

The resources described in this section have been obsoleted. The new information about Hardware Resources in provided in Hardware Resources. The resources described in this section are still supported.

The resource list is the concatenation of two sublists:

- The resources specified with option `-r`, which can be either a static list (explained below) or dynamically computed list as explained in Time-Variant Taskers.
- The resources specified in the `taskerClass.table` file.

The option `-r` in vovtasker defines the resources offered by the tasker.

Examples of explicit resources:

```
% vovtasker -r "unix diskio RAM/512"
% vovtasker -r "RAM/3000 REGRMACHINE"
```

Symbolic Resources for Taskers

The following table describes all symbolic resources defined for taskers. By default, a tasker provides the resources corresponding to the symbolic resource `@STD@`.

Symbolic Resource	Description
@ARCHITECTURE@	It has value 'win64' on Windows, and the same as @MACHINE@ on UNIX.
@CPUS@	A consumable resources indicating the number of CPUS in the tasker (example for a 4-CPU machine: "CPUS/4").
@DISPLAY@	The name of the X display accessible by the tasker and derived from the value of the environment variable DISPLAY.
@HOST@	The name of the host on which the tasker runs.
@MACHINE@	The name that identifies the machine type . On Windows, this have value 'x86'.
@MODEL@	Linux: an abbreviated version of the "model name" line in the output of `/proc/cpuinfo`, otherwise the value is "Model:unknown". Not available on the other platforms.

Symbolic Resource	**Description**
@OS.VERSION@	The name of the operating system, including the version number (for example, Linux.1.0).
@OS@	The operating system name (for example, Linux).
@OSCLASS@	Either "unix" or "windows".

@RAMFREE@	The amount of available physical memory (RAM), in MB. On a Linux system, this is computed by opening `/proc/meminfo` and by adding up the values for MemFree, Buffers, and Cached.
	`% cat /proc/meminfo`
	`MemTotal:       6100392 kB`
	`MemFree:        1848576 kB`
	`Buffers:         188596 kB`
	`Cached:         2686368 kB`
	`SwapCached:          16 kB`
	`Active:         1847404 kB`
	`...`

@RAMTOTAL@	The total amount of physical memory (RAM), in MB.
@RAM@	This is a consumable resource representing the residual amount of RAM after taking into account the jobs running on the tasker.
@RELEASE@	This is mostly used on Linux to represent the name of the Linux distribution. On most systems, this is computed by running `lsb_release -isr`.
@SMARTSUSPEND@	Either the string "smartsuspend" if SmartSuspend (by Jaryba) is available on the machine or the null string "".
@STD@	The default for each tasker, which corresponds to the set @CPUS@ @DISPLAY@ @USER@ @HOST@ @RAM@ @MACHINE@ @VOVARCH@ @OSCLASS@ @OS.VERSION@ @VIEW@ @RELEASE@.
@SWAPFREE@	The amount of available swap space, in MB.
@SWAPTOTAL@	The total amount of swap space, in MB.

Symbolic Resource	Description
@SWAP@	This is a consumable resource representing the residual amount of SWAP after taking into account the jobs running on the tasker.
@USER@	The name of the user that owns the tasker. This is the value of either the variable LOGNAME or of the variable USER.
@VIEW@	The view name as defined by ClearCase.
@VOVARCH@	The value of the environment variable VOVARCH, normally set with `vovarch`.

The taskerClass.table File

This file allows the classification of taskers without requiring the taskers to be restarted. The file `taskerClass.table` resides in the server configuration directory (that is, the directory `$VOV_PROJECT_NAME.swd`). Each tasker checks the file for changes about once a minute.

The file consists of lines with the following format:

```
<taskername>:  <resource list>
(default):  <resource list>
```

All lines that do not conform to this format are ignored. The `taskername` begins at the beginning of the line. If the tasker name is the string `(default)`, the line indicates the default value of the resource list. Note that `(default)` is not a legal tasker name because it contains the characters `()` which are not allowed in tasker names.

```
# Example of taskerClass.table file
(default):  classC

ferrari:  classA classB
buick:    classA
dodge:    classB
```

Time-Variant Taskers

Time-variant taskers resources can be configured using Tcl. Any procedure can be defined in the namespace `VovResources` and then used to compute the resource list.

If the first characters in the argument for the option -r is the sequence `VovResources::`, then the taskers resources are computed by executing the argument as a Tcl command.

For a list of predefined procedures, refer to *Predefined VovResources:: Procedures*.

A special case of a resource is the local disk space on a host, which varies according to the files stored there. There may be jobs that require a minimum amount of space to run successfully. The vovtasker implements `vtk_fs_stat` (see `vtk_fs`) to handle such requirements. An example is provided further below.

An alternative is to use load sensor. However, in the case of free disk space, `vtk_fs_stat` is recommended because it is more efficient.

For example, taskers can be set up to offer one set of resources during the day and another set of resources during the night. The following example is a script that computes the resource "nothing" between 5am and 8pm, and the standard resources during the night. In addition, `vtk_tasker_set_timeleft` is used to control the maximum expected duration of the jobs sent to the tasker.

```
namespace eval VovResources {
    proc Night { args } {
        # Return the standard resources during the night and suspend the tasker
        # during the day.  During the night, the tasker progressively reduces the
        # maximum length of the jobs it accept.
        set NIGHT_RESOURCES  "@STD@ $args"
        set EVENING_START  19
        set MORNING_END     6
        set HH [clock format [clock seconds] -format "%H"]
        regsub {^0} $HH "" HH; # Strip leading 0.
        if { $HH >= $MORNING_END && $HH < $EVENING_START } {
            # -- During the day, suspend the tasker.
            vtk_tasker_set_timeleft  0
        } else {
            # -- During the night, compute the time left.
            set hoursLeft  [expr $MORNING_END - $HH]
            set hoursLeft  [expr $hoursLeft >= 0 ? $hoursLeft : $hoursLeft + 24]
            set timeleft   [expr $hoursLeft * 3600]
            vtk_tasker_set_timeleft $timeleft
        }
        return $NIGHT_RESOURCES
    }
}
```

Example:

```
% vovtasker -r "VovResources::Night hspice"
```

These procedures are evaluated:

- Every minute, or at the interval that was selected with the option -U
- After the completion of each job.

The standard procedures are defined in the script `$VOVDIR/etc/tasker_scripts/taskerRes.tcl`. It is recommended to review these procedures before implementing your own procedures in `$VOVDIR/local/taskerRes.tcl`.

Free Disk Space

The following script monitors free disk space using `vtk_fs_stat`. This example script adds the resources `WORK` and `SCRATCH`; the values of these resources will be the amount of disk space in MB on the corresponding filesystem.

```
namespace eval VovResources {
    proc FsCheck { args } {
        # Usage:
        #   VovResources::FsCheck -fs WORK /work -fs SCRATCH /scratch -r  @STD@ -r xx
        #
        set resources {}
        while { $args != {} } {
            set arg [shift args]
            switch -- $arg {
                "-fs" {
                    set name [shift args]
                    set dir  [shift args]

                    set space [vtk_fs_stat $dir]
                    lappend resources "$name#$space"
                }
                "-r" {
                    lappend resources [shift args]
                }
            }
        }
        return [join $resources]
    }
}
```

Example:

```
% vovtasker -r "VovResources::FsCheck WORK /work"
```

Predefined VovResources:: Procedures

This section describes the predefined procedures that can be used to characterize the taskers in your farm in a time-variant fashion. To use these procedures, you have to use them as resources for a tasker.

An example of using `VovResources::` in `taskers.tcl`:

```
vtk_tasker_define linux102 -resources "VovResources::Offhours -evening 20 -morning
 7"
```

The standard procedures are defined in `$VOVDIR/etc/tasker_scripts/taskerRes.tcl`. You are encouraged to study these procedures and to implement your own procedures in `$VOVDIR/local/taskerRes.tcl`.

Procedure	Arguments	Description
Custom	`args`	Just returns `$args`

Procedure	Arguments	Description
Standard	`args`	Just returns `@STD@ $args`
FsCheck	`args`	This procedure checks the disk space on some filesystems. It is based on the procedure `vtk_fs_stat`. It supports two options: • `-r Resource` to specify a resource of the tasker; it may be repeated • `-fs NAME Directory` to specify the name of a filesystem resource and a directory in that filesystem; it may be repeated Example: `VovResources::FsCheck -fs SCRATCH /export/scratch -r @STD@`
Workstation	`minIdle maxTime args`	This is the old and now obsolete syntax for this policy. Please see below for the updated syntax. This procedure checks the time from the last interaction with the mouse or the keyboard. If it is less than `minIdle`, we assume that the workstation is not idle and we suspend the tasker. If the workstation is idle, we allow jobs with expected duration less than `maxTime`. Arguments `minIdle` and `maxTime` are Time Specifications.
Workstation	`[options]`	Options are: • `-resources "list"` -- Specify the list of resources. • `-minidle time` -- Specify minimum time the tasker has to be idle (no keyboard or mouse activity) for the tasker to become active. • `-maxtime time` -- Specify max expected duration of jobs sent to tasker. • `-suspend` -- By default, when the tasker becomes inactive, the current jobs are completed and no new jobs are accepted. With this option the current jobs are suspended.
Night	`args`	Return the standard resources during the night and suspend the tasker during the day. During the night, the tasker progressively reduces the maximum length of the jobs it accepts.

Procedure	Arguments	Description
Offhours	`args`	Return the standard resources during off hours (night and weekends) and suspend the tasker during the rest of the time. During off-hours, the tasker progressively reduces the maximum length of the jobs it accepts. Supports the following options: *-owner user* To specify a user that has overriding control on this tasker (the user would use `vovtaskerowner`). You also need to specify the -host option. *-host host* **Required if you use -owner**. Specify the name of the host to suspend the jobs when the tasker is suspended. *-susp* To suspend the jobs when the tasker is suspended. *-evening hour* The hour (0-23) at which the tasker starts accepting jobs. The default is 19. *-morning hour* The hour (0-23) at which the tasker stops accepting jobs. The default is 6.

Tasker Reservations

A vovtasker may be reserved for one or more:

- Users
- FairShare groups
- OS groups
- Jobclasses
- Job projects
- Jobs
- Buckets

The reservation is always for a specific period of time, starting at any time and ending some time in the future. If the end time is in the past, the reservation is ignored and removed. The reservation can be very long, for example thousands of days.

A tasker can have multiple reservations at any time. Jobs that do not match any reservation are not sent to the tasker if the tasker is reserved. For example:

- If the tasker is reserved for a user, only jobs submitted by that user may be dispatched to that tasker.
- If the tasker is reserved for a group, only jobs from that group may be dispatched to that tasker.
- If the tasker is reserved for a user and a group, only jobs submitted by that user who belongs to that group may be dispatched to that tasker.

Negated Reservations

Taskers may also be reserved for a negated set, in order to prevent access to the tasker by jobs that match members of the set. To negate a tasker reservation, simply place a not symbol "!" at the beginning of the reservation expression. For example, the command `vovtaskermgr reserve -user '! john,mary' -duration 1h tasker1` will allow the tasker named `tasker1` to run jobs from any user except `john` and `mary` for 1 hour. Any valid reservation expression can be negated.

Note: If you are negating a list, use the not symbol ONLY at the beginning of the list; you should NOT place a "!" before every item.

Persistent Reservations

Reservations are persistent. If you stop a tasker and restart another one with the same name, the new tasker will inherit the persistent reservation of the old tasker. Expired reservations are removed frequently.

If a tasker is renamed, the corresponding reservations get updated too. So, the reservations do not get lost.

Reservation rules

Only reservations with end time later than the current time are valid.
There can be multiple reservations on a vovtasker. But there is only one active reservation.
If there is already an active reservation, a new reservation overlapping time with the active reservation can be created but ignored. When the active reservation expires (end time passes current time), a new active reservation with earliest start time is picked.
Reservation only applies to normal (direct) taskers or BPS agents. Making reservations on indirect taskers is not allowed.

Duplicate reservations

A new reservation can be created if the reservation is different from existing reservations. Two reservations are the same if the following attributes of reservations are the same. All of these attributes can be specified when creating a reservation using `vtk_reservation_create`. If a reservation is created through vovtaskermgr, vtk_tasker_reserve, and `taskers.tcl`, `start time` and `end time` are not used as identifiers. The existing reservation with the same other attributes are updated with the new start time and end time. If there is no existing reservation, a new reservation is created.

`type`	Always `tasker` if the reservation is for tasker.
`what`	List of tasker names that this reservation is reserving
`start time`	Reservation start time
`end time`	Reservation end time
`user`	Reservation is for these users
`group`	Reservation is for these groups
`osgroup`	Reservation is for these osgroups
`jobclass`	Reservation is for these jobclasses
`jobproj`	Reservation is for these jobprojects
`bucketid`	Reservation is for these bucket IDs
`id`	Reservation is for these job IDs

Reserve a vovtasker

There are several ways to create a new reservation.

vovtaskermgr, `taskers.tcl`, and vtk_tasker_reserve creates a new reservation, but if there is an overlapping reservation with the same parameters, the existing one is updated with new `start_time` and `end_time`. These are the same with running `vtk_reservation_create` with the -update option.

On the Command Line: vovtaskermgr reserve

The syntax is:

```
% vovtaskermgr reserve [options] <taskers_list>
```

where the options could be:

-user	Comma separated list of users
-group	Comma separated list of groups
-jobproj	Comma separated list of job projects
-jobclass	Comma separated list of job classes
-osgroup	Comma separated list of OS groups
bucketid	Comma separated list of bucket IDs
-id	Comma separated list of job IDs
-start	Start time

-end	End time
-cancel	
<tasker_list>	List of tasker names to reserve.

For example, the following command reserves taskers `jupiter` and `alpaca` for user john for 3 hours starting from now.

```
% vovtaskermgr reserve -user john -duration 3h jupiter alpaca
```

With `-cancel` option, all reservations on the specified tasker(s) will be removed.

```
% vovtaskermgr reserve -cancel jupiter alpaca
```

The following command shows all reservations for taskers.

```
% vovtaskermgr reserveshow
```

In the taskers.tcl File

This is useful to reserve a tasker from the instant it is created. Use option -reserve of vtk_tasker_define.

The reservation expression argument to the -reserve option takes space-separated list of key value pairs, where the key is one of `USER`, `GROUP`, `JOBCLASS`, `JOBPROJ`, `BUCKET`, `DUR`, `QUANTITY`. If the key is `DUR`, the value is a time spec. If not specified, the default duration is 1 year. For the other keys, the value is a comma-separated list.

Examples:

```
vtk_tasker_define jupitar -reserve "USER john,mary JOBCLASS spectre"
vtk_tasker_define alpaca -reserve "JOBPROJ chipa,chipb DUR 3w"
```

The old form `GROUP/USER/DURATION` is accepted. The `GROUP` and `USER` parts are optional, but the separators ('/', the forward-slash character) must be present. The duration is expressed as a VOV timespec, e.g. 2d for two days. If only digits are present, the value is interpreted as seconds.

-reserve is passed to vovtasker executable as -e option. If advanced users want to run a tasker with -e option for initial reservation, the syntax is the same as `vtk_tasker_define -reserve` option.

Reserving a Tasker via the Browser

Open the tasker page to reserve a particular tasker. It is at the URL `/tasker/taskerId`. Fill out a simple form, indicating which user, group, OS group, job class, and/or job project for this tasker is to be reserved, as well as start time and duration. After you select the duration, the form will be submitted.

Click **Forget** to cancel the tasker reservation, if you are ADMIN.

vtk_tasker_reserve Tcl Interface

With Tcl, you can use tasker reservations. The syntax is:

```
vtk_tasker_reserve taskerId [-user <user1,user2,...>]
                           [-group <group1,group2,...>]
```

```
                    [-osgroup <osgroup1,osgroup2,...>]
                    [-jobclass <jobclass1,jobclass2,...>]
                    [-jobproj <jobproj1,jobproj2,...>]
                    [-bucketid <bucketid1,bucketid2,...>]
                    [-id <jobid1,jobid2,...>]
                    [-start <starttime>]
                    [-end <endtime>]
                    [-duration <reserved_duration>]
```

For example, the following line will clear (cancel) all reservations on tasker 00001230, if there is one. Otherwise, this doesn't have any effect.

```
vtk_tasker_reserve 00001230
```

The following line reserves tasker 00001230 for user `john` for 3 hours starting from now.

```
vtk_tasker_reserve 00001230 -user john -duration 3h
```

This reserves tasker 00001230 for user `john` in group `alpha` for 2 weeks starting from one hour from now.

```
vtk_tasker_reserve 00001230 -user john  -group alpha  -start [clock scan "1 hour"]  -
duration 2w
```

vtk_reservation_create Tcl Interface

You can use `vtk_reservation_create` in a Tcl interface. This is a new interface introduced in 2017 version to support multiple reservations per tasker.

The syntax is:

```
vtk_reservation_create <type> <what> <quantity> <start_time> <end_time> [options]
```

where:

<what>	Comma separated list of tasker names
<quantity>	Not used for tasker reservations
<start_time>	Reservation start time
<end_time>	Reservation end time

And the options could be:

-user	Comma separated list of users
-group	Comma separated list of groups
-osgroup	Comma separated list of OS groups
-jobclass	Comma separated list of jobclasses
bucketid	Comma separated list of bucket IDs
-id	Comma separated list of job IDs

-update

For example, the following Tcl script creates a reservation for host1 and host2. It reserves 4 slots of each host. It returns ID for the reservation. The ID can be used to update and delete the reservation.

```
set now [clock seconds]
vtk_reservation_create tasker host1,host2 1 $now [expr $now+3600] -user brian
```

If all attributes are the same as one of existing reservations, `vtk_reservation_create` will return `nochange`.

With -update option, it looks for a reservation which has same attributes except `start_time` and `end_time` but the reservation period is overlapping. If there is one, the existing reservation is updated with `start_time` and `end_time`.

```
set now [clock seconds]
set end [expr $now+3600]
set end2 [expr $now+7200]
vtk_reservation_create tasker localhost 1 $now $end -user john #creates a new
 reservation
vtk_reservation_create tasker localhost 1 $now $end -user john #returns "nochange"
vtk_reservation_create tasker localhost 1 $now $end -group g1  #creates a new
 reservation
vtk_reservation_create tasker localhost 1 $now $end2 -user john -update #updates the
 first
```

Manage Reservations

To show all existing reservations, use:

```
% vovshow -reservations
```

To forget all reservations, use:

```
% vovforget -allreservations
% vovforget <reservationId>
```

Reservations data is accessible with `vovselect` as well.

```
% vovselect id,reserveuser,reservestart,reserveend from reservations
```

vtk_reservation_get Tcl Interface

`vtk_reservation_get <reservationId> <variable>` provides the details about a reservation.

```
vtk_reservation_get 21673 info
parray info
```

Prints out the following information:

```
info(id)               = 21673
info(quantity)         = 1
info(reservebucketid) =
info(reservecreated)  = 1513026518
info(reservedby)       = jin
info(reserveend)       = 1513199318
info(reservegroup)     =
info(reserveid)        =
info(reservejobclass) =
info(reservejobproj)  =
info(reserveosgroup)  =
info(reservestart)     = 1513026518
info(reserveuser)      = jin
info(type)             = tasker
info(what)             = local2
```

`vtk_reservation_update <reservationId> <fieldname> <new_value>` updates a field of reservation with a new value. Available field names can be found by `vovselect fieldname from reservations` on the command line.

```
vtk_reservation_update 21673 RESERVEUSER robert
```

`vtk_reservation_delete <reservationId>` removes the reservation.

```
vtk_reservation_delete 21673
```

Number of Reservations and System Performance

Many reservations on each tasker may slow down the system. Upon choosing the right tasker to run a job, the algorithm considers all reservations. It is recommended to use the limited number of reservations per tasker. By default, the maximum number of reservations per tasker is set as 10 and this is configurable through a server parameter *`tasker.max.reserve`* in `policy.tcl`.

Tasker Owner

The program `vovtaskerowner` gives the owner of a machine the control on the tasker running on such machine.

By invoking `% nc cmd vovtaskerowner &` you get the dialog shown below.

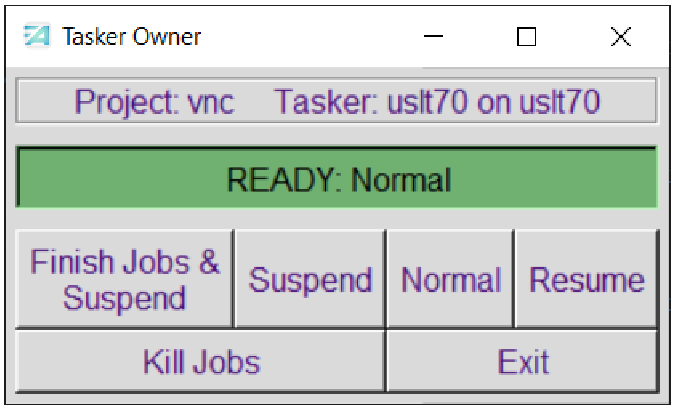


Figure 3:

Suspend suspends the tasker immediately for one hour. Each time Suspend is clicked, the tasker is suspended for an additional hour.

Resume resumes the activity of a suspended tasker for an hour. Each time Resume is clicked, the tasker remains active for an additional hour.

Normal returns the tasker to the normal behavior.

This dialog maintain its status in a file called `~/.vnc_<TASKERNAME>`. This is also the file that is being read by the vovtasker binary.

Usage: vovtaskerowner -h

```
vovtaskerowner: Usage Message

  DESCRIPTION:
      vovtaskerowner [OPTIONS]

  OPTIONS:
      -v                 -- Increase verbosity
      -h                 -- This help
      -taskername  NAME  -- Specify the  name of the tasker we want to control.

  EXAMPLES:
      % vovtaskerowner -h
      % vovtaskerowner &
      % vovtaskerowner -taskername mac05 &
```

Administrator Reponisibilities

The mechanism shown in this page works only if the tasker is using the time-variant resources "VovResources::Offhours"

```
# Fragment of taskers.tcl file.
# This line allows user johnny to be considered the "Owner" of
# the tasker on machine linux123
vtk_tasker_define linux123 -resources "VovResources::Offhours -owner johnny"
```

Start a Remote UNIX Tasker Details

On UNIX, the script `vovtaskerstartup` is used to start a tasker on a remote machine.

This script ensures that the tasker runs in a valid environment by sourcing the following scripts:

- `$VOVDIR/local/scripts/vovtaskerstartup.aux`, if available, to perform site specific initialization:

  ```
  # Example of $VOVDIR/local/scripts/vovtaskerstartup.aux
  echo "This is vovtaskerstartup.aux on `uname -n`"
  switch ( $VOVARCH )
   case "linux*":
     unlimit openfiles
     breaksw
   default:
  endsw
  ```

- `$VOVDIR/etc/std.vov.aliases`, to define all standard aliases (for example, `ves`)

- `setup.tcl` in the server configuration directory to initialize the project environment.

If the -view option is used, the script also starts the tasker in the given ClearCase view.

The -descriptors option triggers a check to ensure the host has at least as many descriptors as vovserver, which ensures it can operate at full capacity in the event of failover. It also checks that the `server_election` directory is empty.

Note: For this function to work, the -failover option must be used with vtk_tasker_define.

Stop Taskers

When you no longer need a tasker that you started manually and that is running in the foreground, stop it with `Ctrl-C`.

Background taskers, and taskers started with `vovtaskermgr`, can be shut down forcibly with `vovkill` *`pid`* or more cleanly with

```
% vovtaskermgr stop
% vovtaskermgr stop <tasker_name>
```

Clean Up

Occasionally, the taskers leave some files in `/usr/tmp`. You can use `vovcleanup -host` to browse for files left over by the taskers.

```
% vovcleanup -host
```

Interfaces to Accelerator, SGE, LSF, and Others

FlowTracer can be used also as a front end to an existing batch processing system (BPS), for example, Accelerator, LSF, SGE, SGEEE, etc.

Connect to a Third Party Batch Processing System

FlowTracer provides default implementations of interfaces for some of the Batch Processing Systems. FlowTracer also provides Tcl APIs for easy, flexible and extensible implementation of interfaces to the Batch Processing System of your interest. This enables FlowTracer to fit into any existing Enterprise Grid.

Architecture Diagram

The following diagram shows how FlowTracer projects work with several Batch Processing Systems. In this example, they are Accelerator and SGE.

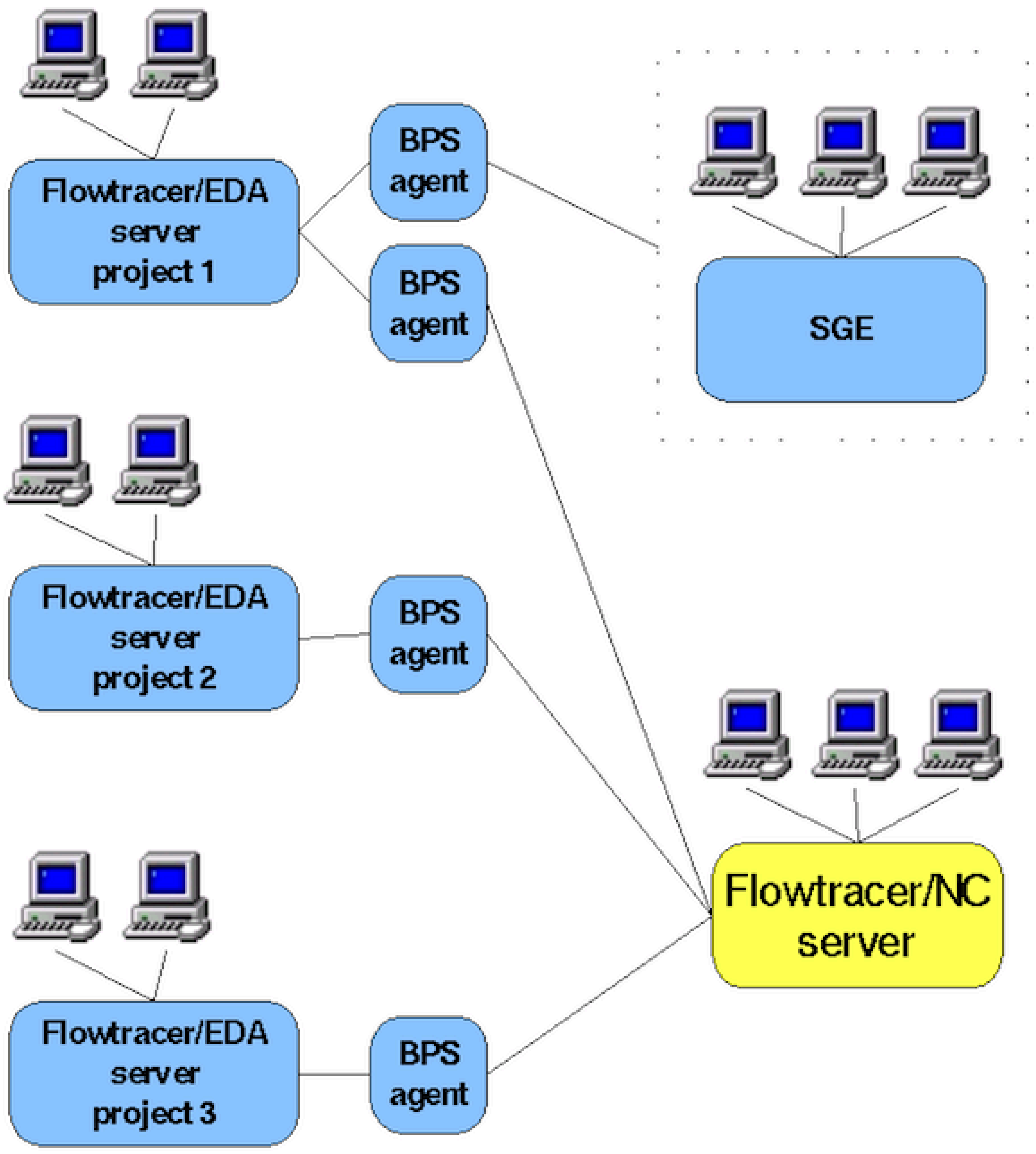

Flowtracer/EDA server project 1
BPS agent
BPS agent
SGE
Flowtracer/EDA server project 2
BPS agent
Flowtracer/NC server
Flowtracer/EDA server project 3
BPS agent

Figure 4:

Batch Processing System Agent

You can create a Batch Processing System agent as seen in the diagram as the **BPS agent**. It is the interface that submits jobs to the Batch Processing System that it gets from a FlowTracer server, instead of executing them directly.

Interface FlowTracer with Accelerator

There are two ways to interface FlowTracer to Accelerator.

They are:

- With a **binary interface** capable of supporting multiple users. The jobs are injected directly into the memory of the Accelerator server, thus preserving safely the credentials (user, osgroup) of each job. This method bypasses the policy layers implemented as part of the `nc run` command, which may be a disadvantage in some cases.
- With a **scripted interface** limited to just one user, namely the owner of the FlowTracer project. This mode is similar to the one used to interface to LSF or SGE. The advantage of this approach is that the same `nc run` interface is used to submit the jobs, so all jobs are processed by all policy layer implemented as part of `nc run`.

Using the Binary Interface

FlowTracer provides an optimized implementation in C++ for the interface to Accelerator.

As discussed in Connect to a Third Party Batch Processing System, a Batch Processing System agent (BPS agent) is a vovtasker that connects to your FlowTracer vovserver and dispatches jobs to a backend Batch Processing System. You can start a BPS agent vovtasker from the command line by using the -V option of the `vovtasker` command, or you can define a BPS agent to Accelerator in `taskers.tcl` so that the BPS agent will be started when you start the FlowTracer project.

Start a BPS Agent from Command Line

First, decide which Accelerator to use to send jobs to which Accelerator project. Usually the Accelerator queue is called 'vnc', but you can find the available ones by asking your administrator, or by using:

```
% vovproject list -all -l | grep vnc
```

The command line syntax to start a BPS agent is:

```
% vovtasker -V qname@host[:port]
```

where `port` is optional and can be omitted if the port number of the Accelerator server is the default one for the queue name.

For example, an Accelerator server is running with the default name `vnc`, and is running on host `alpaca` with the default port `6271`. You can connect a BPS agent to the FlowTracer project and send jobs to this Accelerator server by:

```
% vovtasker -V vnc@alpaca                # or
% vovtasker -V vnc@alpaca:6271
```

You can use other options of `vovtasker` to specify other attributes of your BPS agent. Among all the attributes, the default value of "capacity" for BPS agent is 100 while default for normal tasker is equal to the number of tasker machine's CPUs. You can override this default using the -T option. Usually you want to have a large capacity, because you want to give the BPS the ability to schedule jobs in the most effective manner. Capacities of 200 or 300 are typical.

Define a BPS Agent in taskers.tcl

In the `taskers.tcl` configuration file, you use `vtk_tasker_nc` to define a BPS agent to Accelerator.

After you define the BPS agent in `taskers.tcl`, it will be started when you restart your FlowTracer project. You can start it without restarting your project by using the command

```
% vovtaskermgr start name-of-BPS-agent
```

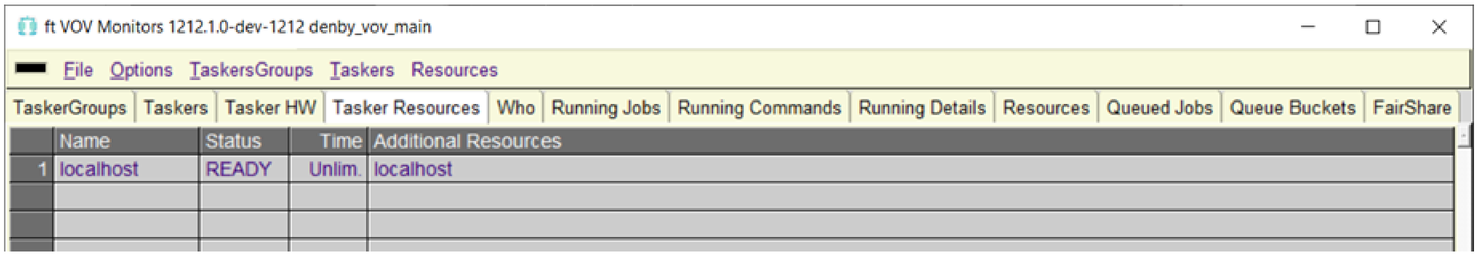
Unless you give it a different name, the name of the BPS agent will be `NC`. See instructions for The taskers.tcl File for details.

Examples

Here is an example, where in the context of FlowTracer project `test1@alpaca`, we now start a BPS agent which interfaces to FlowTracer `vnc@alpaca`:

```
% vovtasker -V vnc@alpaca -a NC -r "se_license spice_license"
```

Here's what the tasker monitors look like:


ft VOV Monitors 1212.1.0-dev-1212 denby_vov_main
File Options TaskersGroups Taskers Resources
TaskerGroups Taskers Tasker HW Tasker Resources Who Running Jobs Running Commands Running Details Resources Queued Jobs Queue Buckets FairShare

	Name	Status	Time	Additional Resources
1	localhost	READY	Unlim.	localhost

Figure 5:

Here is the architecture diagram in this case:

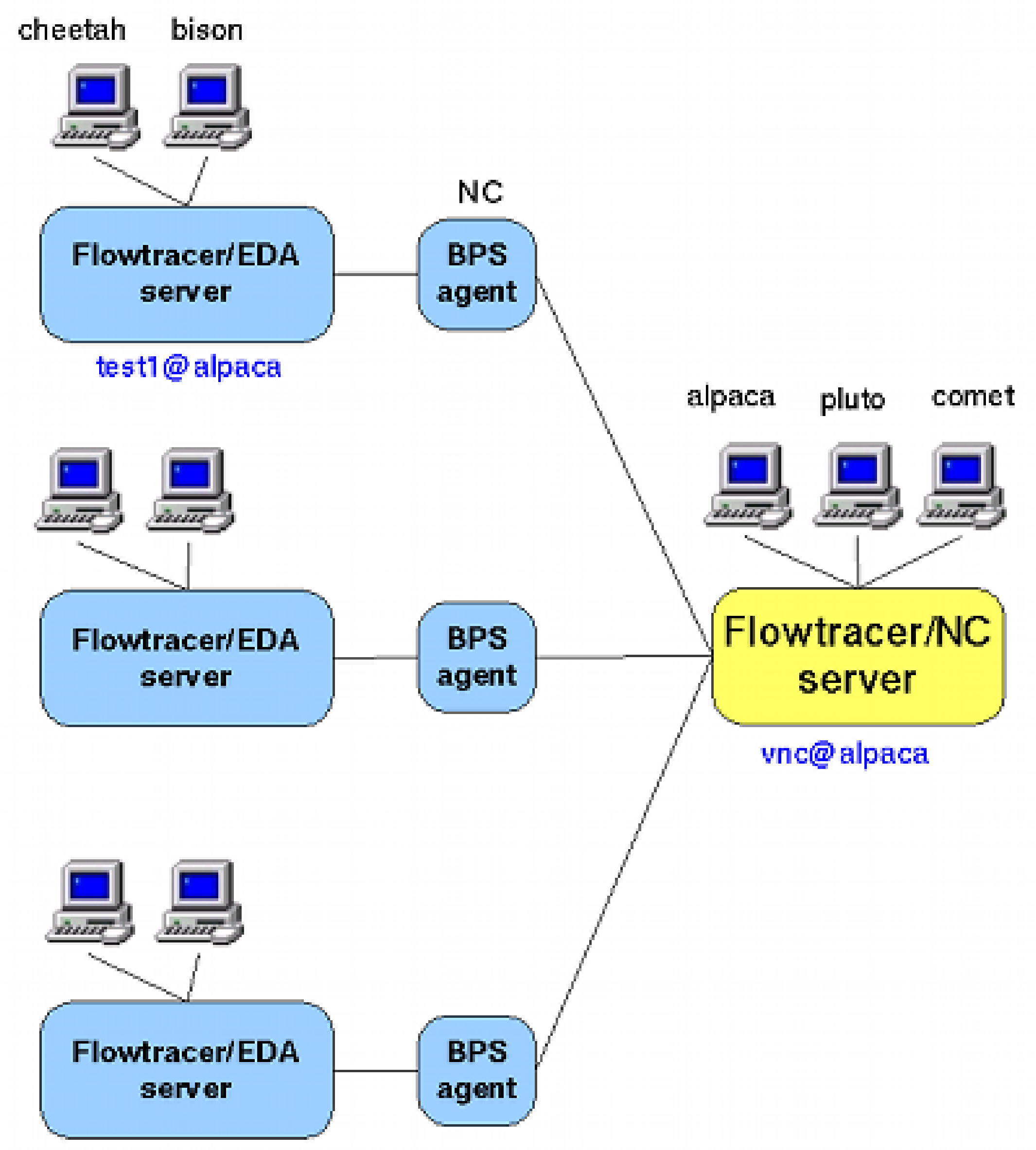


Figure 6:

The above example is a FlowTracer project named `test1@alpaca`. There are two normal "direct" taskers `cheetah` and `bison` already connected to it. The Accelerator server `vnc@alpaca` has 3 taskers, `alpaca`, `pluto` and `comet`. A BPS agent named `NC` is started in the context of `test1@alpaca`, which connects it to Accelerator `vnc@alpaca`. The taskers of `vnc@alpaca` are represented as `BPS taskers` prefixed with the name of the `BPS agent` in `test1@alpaca` for easy monitoring and management.

Handling Resources

The server managing the FlowTracer project uses its resource maps to determine which vovtasker should receive each job. When a job requests a floating resource, for example a license, that request needs to be passed through the BPS agent to Accelerator, so the job will not be started when the license is unavailable.

There are several methods you can use so that the jobs will run correctly in Accelerator and be dispatched through the BPS agent.

- If the number of resources is small, you can declare them using the -resources option of `vtk_tasker_nc`, or the -r option with the -V option of vovtasker on its command line.
- When the jobs in your FlowTracer project need many different resources from Accelerator, you can use the `ncconfig.tcl` file with the -F option of vovtasker for the BPS agent.

  You can start the BPS agent vovtasker

  - with the `vovtaskermgr` command
  - automatically with a `liveness` script
  - manually from the command line

Here is an example of the interface configuration file between FlowTracer and Accelerator.

```
# Interface control file for NC dispatcher (-V) vovtasker for an FT project
#parray NCTASKER; # uncomment for troubleshooting

# Integer, controls debug messages
set NCTASKER(flag,debug) 0

# int, whether to show the farm tasker LEDs in the FT GUI
set NCTASKER(flag,show,taskers) 0

# BPS agent offers resources of this type from NC, e.g License:PrimeTime-SI
set NCTASKER(tasker,resources,resmap,types,export)  {License}
# set NCTASKER(tasker,resources,resmap,types,export) {Priority}

# These are offered by the BPS agent in addition to the exported res from NC
set NCTASKER(tasker,resources)       "NC linux sun7"

# set NCTASKER(job,resources,start) ""

# Resources to delete from jobs dispatched to NC
# set NCTASKER(job,resources,del)   ""

# Resources to add to jobs dispatched to NC
# set NCTASKER(job,resources,add)   ""

# set NCTASKER(job,group)           ""
```

You can place this file in any convenient directory, but it is recommended to place it in the `.swd` of the FlowTracer project. This directory must be accessible to the FlowTracer project anyway, and the `live_start_nctasker.tcl` script expects it to be there.

When using the interface file with BPS taskers defined in `taskers.tcl` there is a shortcut that causes the file to be searched for first in the `.swd` of the `FT` project, and then in `$VOVDIR/local`.

To setup your FlowTracer project so that an Accelerator BPS agent can be managed with the `vovtaskermgr` command, use the -nccfg option of `vtk_tasker_nc`, for example:

```
vtk_tasker_nc -name NCbps -nccfg path-to-ncconfig.tcl
```

Note: If you use the special value **-** (minus sign) for `path-to-ncconfig.tcl`, the search mechanism described above will be invoked. Whether by search or path specification, the config file must be found or the BPS vovtasker will not be started.

To manually start a BPS agent vovtasker from the command line with the interface configuration file, use:

```
% vovtasker -V qname@host[:port] -F path-to-ncconfig.tcl
```

To setup your FlowTracer project so that an Accelerator BPS agent will start automatically whenever jobs need to be run in your FlowTracer project, set up the `ncconfig.tcl` file in your FlowTracer project's `.swd` directory. This is the only place where the standard liveness script looks for the interface file.

Also, copy the example liveness script to the `tasks` subdirectory of your FlowTracer project's `.swd` configuration directory. You may need to make the `tasks` directory there.

```
%  cp $VOVDIR/etc/liveness/live_start_nctasker.tcl `vovserverdir -p tasks`
```

To test whether it is working, retrace some jobs from your FlowTracer project. You may need to modify the jobs so that they request the resource 'NC' that is offered by the BPS dispatcher. They should become SCHEDULED (light blue color). A new vovtasker should appear, then the job will become dispatched (cream color) and then RUNNING (yellow), and finally VALID or FAILED.

Check the resources offered by the BPS dispatcher tasker using the monitors, the browser, or by

```
% vovtaskermgr show -resources
```

The log file for the BPS dispatcher is in the usual place for tasker logs, in the directory `.swd/logs/taskers/the-tasker-name`.

Using the Scripted Interface

These taskers can be declared in the same way as the binary interface, except that you have to use the option -singleUser 1 in `vtk_tasker_nc`. The script `$VOVDIR/etc/tasker_scripts/taskersVNC.tcl` provides an interface to Accelerator.

Interfaces to Other Batch Processing Systems

You can find examples of interfaces in the directory `$VOVDIR/etc/tasker_scripts`. The file `$VOVDIR/etc/tasker_scripts/taskerLSF.tcl` implements the interface to LSF.

Use Implemented Interface to SGE

You can use the -I option of `vovtasker` to start a BPS agent. For example, FlowTracer implements the interface to SGE in `$VOVDIR/etc/tasker_scripts/taskerSGE.tcl`. You can start the agent to SGE like this:

```
% vovtasker -I $VOVDIR/etc/tasker_scripts/taskerSGE.tcl -T 300 -r "dc_shell_license
 dracula_license"
```

You can use other options of `vovtasker` to specify other attributes of your BPS agent. Among all the attributes, the default value of "capacity" for BPS agent is set to 100 (while the default for a normal tasker is equal to the number of tasker machine's CPUs). You can always overwrite this default by -T option. Usually you want to have a large capacity, because you want to give the BPS the ability to schedule jobs in the most effective manner. Capacities of 200 or 300 are typical.

Study and Write Your Own Implementation to Other BPS

The BPS agent is implemented in a BPS agent file, which is a Tcl script that implements the procedures described in the table below.

To implement the BPS agent for your BPS system, for example, LSF, you just need to implement the procedures in the following table. You are encouraged to use the provided implemented interface to SGE as example.

Procedure	Description
`taskerStartJob`	This procedure submits a job to the BPS. All information about the job is in the global array `jobDesc`. It should return a reference id, i.e. the id of the job in the context of the BPS.
`taskerStopJob`	This procedure is called when the tasker wants to stop a running or queued job.
`taskerCheckJob`	This procedure is called when the tasker wants to check the status of a job. It returns one of the following values: LOST DONE FAILED RUNNING QUEUED.
`taskerResumeJob`	This procedure is called when the tasker wants to resumed a suspended job.
`taskerSuspendJob`	This procedure is called when the tasker wants to suspend a job.
`taskerJobEnded`	This procedure is called when the tasker receives a notification from the server that a job has ended. This allows the tasker to update its state promptly instead of waiting for another timeout.
`taskerMapResources`	A procedure to map the resources required by the job in the context of the local project and the resources required by the job in the context of the BPS.
`taskerCleanup`	This procedure is called when the indirect tasker exits. It should be used to cleanup the garbage that may have been created by the tasker.

taskerStartJob

The procedure `tasker Start` takes a single argument, the `jobId`. The rest of the job information is available in the array jobDesc, as described in the following table:

Variable Name	Meaning
jobDesc(command)	The complete command line.
jobDesc(env)	The environment label for the job.
jobDesc(id)	The job Id.
jobDesc(priority)	The VOV priority level.
jobDesc(resources)	The resource list for the job.
jobDesc(user)	The user that owns the job.
jobDesc(xdur)	The expected duration.

The procedure returns the reference ID used by the BPS. Example:

```
proc taskerStartJob { jobId } {
 global jobDesc env
 # Generate a label from the command by eliminating
 # all non alphanumeric characters.
 set label $jobDesc(command)
 regsub -all {[^a-zA-Z0-9_]+} $label "_" label
 set label [string range $label 0 7]
 set submitInfo [exec qsub -V -v VOV_ENV=BASE -j y -N $label  $env(VOVDIR)/scripts/
vovfire $jobId]
 set refId  [lindex $submitInfo 2]
 return $refId
}
```

taskerStopJob

This procedure is called when the tasker wants to stop a job. The procedure takes two arguments: the `VovId` of the job and the `referenceId` returned by `taskerStartJob`.

Example:

```
proc taskerStopJob { jobId refId } {
    # Stop a SGE job.
    exec qdel $refId
}
```

taskerCheckJob

This procedure takes two arguments: the `VovId` of the job and the referenceId returned by `taskerStart`. It is called when the tasker wants to find out the status of a job. The procedure is expected to return one of the following values:

Return Value	Meaning
LOST	The job is no longer in the BPS. It is generally assumed that the job is done.
DONE	The job is done.

Return Value	Meaning
FAILED	The BPS believes that the job has failed.
RUNNING	The job is currently executing.
QUEUED	The job is in the BPS queue.

Example:

```
proc taskerCheckJob { jobId refId } {
    # Check status of a SGE job.
    set status [ParseOutputOf [exec qstat] $refId]
    if { $status == "RUNNING" } {
        vtk_tasker_job_started $jobId [GetStartTime $refId]
    }
    return $status
}
```

If the job is running, it is the responsibility of this routine to inform the tasker of the exact time the job was started by invoking:

```
vtk_tasker_job_started $jobId $timespec
```

taskerSuspendJob, taskerResumeJob

These procedures are used to suspend and resume a job. These procedures also take two parameters: the *jobId* and the *referenceId*.

taskerJobEnded

This procedure is called when the tasker receives a notification from the server that a job has ended. This allows the tasker to update its state promptly instead of waiting for another timeout.

taskerCleanup

This procedure is called when the indirect tasker exits. It should be used to cleanup the garbage that may have been created by the tasker.

Subordinate Resources

The subordinate resources are the resources of a job that appear after the first '--' in the resource specification.

For example, in the resource specification:

```
JobType:compile -- ( alpaca OR sun7 )
```

The subordinate resources is the string `(alpaca OR sun)`

These subordinate resources are used by indirect taskers in the submission of the job to the remote BPS.

In the following example, the primary resource is `NC`, which is a simple way to route a job towards an indirect tasker connected to Accelerator, while `License:drc1` OR `License:drc2` is the subordinate resource which will be used when the job is transferred to Accelerator.

```
R "NC -- License:drc1 OR License:drc2"
J vw run_some_drc Chip.x
```

The field to access the subordinate resources is called "SUBRESOURCES".

Interface with LSF using taskerLSF.tcl

Start the LSF Tasker

You can start the tasker from the command line:

```
% vovtasker -a LSF -i 0 -I $VOVDIR/etc/tasker_scripts/taskerLSF.tcl -r "lsf" -T 100
```

Alternatively, you can define the tasker in the `taskers.tcl` file:

```
# Fragment of taskers.tcl file
vtk_tasker_define localhost -name LSF  -indirect  $env(VOVDIR)/
etc/tasker_scripts/taskerLSF.tcl  -transient 0
-resources "lsf"  -capacity  100
```

The tasker is **non transient**, to preserve the status of the jobs submitted to LSF even in the event of a crash of the tasker.

Environment Variable VOV_LSF_QUEUES

The variable VOV_LSF_QUEUES is a comma separated list of queue names. If the variable is defined, the first queue in the list is going to be the default queue to which the jobs are submitted. This default can be overridden on a job by job basis by defining the resource "Queue:QNAME".

Job Submission with vovfire

Each job submitted from `taskerLSF.tcl` uses `vovfire`. The submission command is going to have the following form:

```
bsub [BSUB OPTIONS] vovfire 00012345 label_computed_from_command_line
```

Resource Mapping

The procedure `taskerMapResources` in file `taskerLSF.tcl` performs mapping of job resources into LSF submission specifications. This procedure can be redefined in a file called `$VOVDIR/local/taskerLSF.tcl`. The default mapping is described in the following table.

Job Resource	LSF bsub parameter	Example	
Jobname:jname	-J jname	Jobname:abc	-J abc
LSFcpus#N	-n $N	LSFcpus#4	-n 4

Job Resource	**LSF bsub parameter**	**Example**	
LSFtiles#N	-n $N	LSFtiles#2	-n 2
License:lic	rusage [lic=1]	License:qx	rusage [qx=1]
Queue:qname	-q qname	Queue:night	-p night
RAM/*	rusage [mem=*]	RAM/200	rusage [mem=200]
RAMFREE/*	rusage [mem=*]	RAMFREE/200	rusage [mem=200]
swp/*	rusage[swp=*]	swp/100	rusage [swp=100]
tmp/*	rusage[tmp=*]	tmp/200	rusage [tmp=100]

anything else is ignored

Configuration for Elastic Computing

vovwxd

`vovwxd` is a daemon used for dynamically requesting taskers from another job scheduler, that is, a base queue. This process is referred to as *elastic computing*. This daemon is the key component of the Accelerator Plus product and is also used with FlowTracer when interfacing with an external job scheduler.

In general, `vovwxd` connects to a base queue and watches the local Accelerator Plus or FlowTracer workload; if a job bucket is waiting for hardware or software resources, `vovwxd` issues a request to the base queue for a tasker that will provide more resources to the local scheduler. This request is referred to as a *launcher job*. The launcher job runs on a local system tasker running under the name `WXLauncher` and is simply a job submission command to the base scheduler for the `wxagent` utility. As long as the base queue provides the resources the job is waiting for, the `wxagent` utility will run in the base queue and will spawn a vovtasker that connects back to the local vovserver. The workload is executed as the base scheduler satisfies the requests for more taskers from `vovwxd`. Typically, the taskers requested by `vovwxd` offer only one slot, so that only one job at a time can be run on one of these taskers.

The daemon is normally started automatically via the `SWD/autostart/start_vovwxd.tcl` script and is managed on-demand with the `vovdaemonmgr` utility. The daemon can be executed as a foreground process for debugging purposes.

Configuration

In order to successfully run jobs in an elastic computing environment, some configuration must be done up front. The required files are `config.tcl`, and a base-queue-specific driver script, such as `vovnc.tcl` for Accelerator. Both files reside in the `SWD/vovwxd` directory. To install these files with an initial configuration, use the `vovwxconnect` utility. The utility provides options that configure the base queue type and name. Any change to the `config.tcl` file or the driver script after product start is automatically picked up by `vovwxd`.

The `WXLauncher` tasker is automatically configured and started when connecting to Accelerator Plus queues via `vovwxconnect -wx`.

In Accelerator

From the Accelerator perspective, you can look at a set called "WXTaskers:wx" to see all requested jobs coming from `vovwxd`. Each set should have some number of valid jobs (taskers that have exited normally) and few running/orange jobs (the current work being done) and a small number of queued/cyan jobs. Only a small number of queued jobs should ever be present (e.g. 10-50) even though the Accelerator Plus session may have several hundred thousand jobs ready to run. The `vovwxd` daemon will continue to replenish the pending number as they get executed, so that the Accelerator queue size remains small. This does not impact FairShare negatively as only a single waiting job causes FairShare to kick in; teams that use Accelerator Plus will not be at a disadvantage just because a handful of jobs are present in Accelerator at any one time compared with others that run their entire workload in Accelerator.

Migrating from 2016.09 Version

When elastic computing was introduced in version 2016.09, it was based on Tcl daemons called `vovelasticd` and `vovlsfd`. This has been updated to a new process based on the binary daemon `vovwxd`.

Upon starting, `vovwxd` checks the `WXSWD/autostart` directory for scripts to start legacy daemons, such as `vovelasticd` or `vovlsfd`. If such legacy autostart scripts exist, `vovwxd` prints a warning stating that the `vovwxd` autostart script will not be installed. The legacy daemon will need to be manually stopped, and its autostart script will need to be disabled (renamed or removed). The `vovwxd` autostart script will then need to be copied from `$VOVDIR/etc/autostart` into `WXSWD/autostart`. If a legacy daemon autostart script does not exist, the autostart script for the new `vovwxd` daemon will be installed into `WXSWD/autostart` and the daemon will be automatically started.

Three environment variables can be set:

VOV_WX_STUCK_VOVWXD_AGE
: Age at which the `vovwxd` daemon should be considered as stuck and an alert should be generated.

  Set in `wx.swd/setup.tcl`. Age can be in timespec or seconds format.

VOV_WX_STUCK_VOVWXD_RESTART
: If set to 1, the `vovwxd` daemon will be restarted if the stuck age has been exceeded.

VOV_WX_STUCK_VOVWXD_ALERT_AGE
: If set to 1, the `vovwxd` daemon will be straced if the stuck age has been exceeded.

vovwxd Process Loop

The `vovwxd` daemon performs the following steps in its main loop.

Note: All Tcl procedures mentioned below must be defined in the driver script.

1. Reload `config.tcl` or `vov*.tcl` files if they've been modified.
2. Call the `OnConfigLoaded` Tcl procedure.
3. Process existing launcher jobs.

Option	Description
If a job is VALID	Create a tasker object. Calls `ParseLauncherOutput` Tcl procedure.
If a job is FAILED	Report the error (`vovnc` will check if the same error occurred a 2nd time). Calls `OnLaunchError` Tcl procedure.
If a job is RUNNING	Report a warning if the age is >= 30s

Option	Description
For all other options	Delete the job.

4. Delete the LJO (launcher job output) file from the `SWD/vovwxd/launchers` directory.
5. Obtain list of all taskers and buckets along with their metadata (job counts, users, tasker states, etc.).
   a) Call `OnProcessBucket` Tcl procedure for new bucket(s), where the driver script should populate the LAUNCHINFO array. The most important array keys are:

```
set LAUNCHINFO(cmd,submit)     nc run -q .... (the only mandatory key)
set LAUNCHINFO(cmd,submitArgs) appends additional args to submit command
set LAUNCHINFO(tasker,msg)      "Queued in NC queue <name>"
set LAUNCHINFO(taskerRes)       vovtasker's -r value
set LAUNCHINFO(maxlife)        vovtasker's -Z value
set LAUNCHINFO(maxidle)        vovtasker's -z value
set LAUNCHINFO(submit,jobName) "taskerFor_$LAUNCHINFO(BUCKETID)"
```

6. Decide how many new vovtasker are needed for each bucket based on existing vovtasker count and limits config parameter.
   a) Call `OnAdjustNumLaunchers` Tcl procedure, that can fill in the following array fields:

```
set LAUNCHINFO(submit,numJobs)       0 (mandatory)
set LAUNCHINFO(submit,numTaskers)     0 (mandatory)
set LAUNCHINFO(submit,skipReason)    "Skipped because no working queue
 found."
set LAUNCHINFO(limit,maxReached)     1
set LAUNCHINFO(submit,taskerLogPath) "specifies full log file path for
 each tasker"
```

7. Create new launcher jobs.
8. Check for the vovtaskers that are not connected.

Option	Description
If the vovtasker was sick for more than CONFIG(sick,older) seconds	Kill it
If the vovtasker has never connected	Check the `wxagent` job status. Calls the `GetStatusOfJobList` Tcl procedure. • If the job is queued or running, no action (waiting on a hardware resource). • If the job has failed to start, delete the vovtasker object. • The `wxagent` job can be deleted in the base queue, delete the vovtasker object • The base queue may not be accessible, delete the vovtasker object.

Option	Description

9. Start a new log cleaner thread based on `CONFIG(launchers,checkfreq)`.

vovwxd Daemon Configuration Parameters

set CONFIG(driver_script)
: `vovnc.tcl`
: Tcl script that provides the `vovwxd` daemon with the information on integrating with a base queue. Accelerator (`vovnc.tcl`) and PBS (`vovpbs.tcl`) are provided and supported.

set CONFIG(queues)
: vnc
: A space-separated list of base queues to use for submitting tasker requests. Used for Accelerator base queues only.

set CONFIG(cmd,env)
: BASE+D(VOV_UMASK=022)
: A default named environment to use for the launcher job that will submit a tasker request to a base queue.

set CONFIG(cmd,submit)
: ""
: Overrides the base command the launcher job will call (that is, `nc run`, not normally used)

set CONFIG(jobclass,resourcemode)
: wx
: Specifies whether to process job class scripts in Accelerator Plus (wx) only or in the Accelerator base queue(s) as well. If set to "nc", the job class scripts must be authored with duplicate resource request protection measures. Used for Accelerator base queues only.

set CONFIG(refresh)
: 5s
: The `vovwxd` loop cycle. If the loop time is less than this value, it will wait until this much time has passed before starting the next cycle. If the loop time is more than this value, the next cycle will begin immediately after the current one is finished.

set CONFIG(jobstat,checkfreq)
: 5m
: The frequency of which `vovwxd` will check for tasker job status in the base queue(s).

set CONFIG(sick,older)
: 5m
: If a tasker goes sick, it will be removed if it is still in that state after this much time has passed.

set CONFIG(tasker,max)
: 99

The maximum number of taskers that are allowed to be managed by the Accelerator Plus queue at any given time.

set CONFIG(tasker,maxtaskersPerBucketPerNCQueue)
"10"
The maximum number of taskers that are allowed to be requested for each Accelerator Plus queue bucket, per base queue.
Example: 3 base queues would result in 30 taskers being requested in total, at a maximum. Used for Accelerator base queues only.

set CONFIG(tasker,minQueuedPerBucketPerNCQueue)
"1"
The minimum number of taskers that are allowed to be requested for each Accelerator Plusqueue bucket, per base queue.
Example: 3 base queues would result in 3 taskers being requested in total, at a minimum. Used for Accelerator base queues only.

set CONFIG(tasker,arrayMax)
0
When there is enough workload to require multiple taskers from a base queue, request them as a single job array instead of individually submitted jobs. A value of 0 disables this feature, while any positive number will enable it.

set CONFIG(tasker,maxlife)
1w
Taskers are reused as long as there are more jobs in the bucket that originated the tasker request. This parameter limits how long will accept jobs from the bucket. It is important to remember that in the base queue, this would be a single, week-long job (given there are enough jobs in the bucket to keep it going) and therefore will have an impact on FairShare in the base queue.

set CONFIG(tasker,maxidle)
30s
This is the maximum amount of time a tasker will wait for a new job from the same bucket before exiting.

set CONFIG(tasker,res)
""
Specifies additional resources to be provided by taskers. Not normally used.

set CONFIG(tasker,update)
15s
Specifies the vovtasker's update interval for resources (ie, the time between updating the vovserver with resource usage data).

set CONFIG(tasker,debug)
0
Set to 1 to enable the printing of debug messages in the tasker log file.

set CONFIG(log,level)
6

Controls the logging verbosity of the `vovwxd` daemon. Range is from 0 (off) to 10 (extremely verbose, for debug purposes).

set CONFIG(log,path)

""

Overrides the default `vovwxd` log file path (`SWD/vovwxd`).

set CONFIG(launchers,autoForgetSuccessful)

1m

Specifies how long to keep successfully executed launcher jobs in the system's memory.
Enabled only when log,level is 10.

set CONFIG(launchers,autoForgetFailed)

3d

Specifies how long to keep failed launcher jobs in the system's memory.
Enabled only when log,level is 10.

set CONFIG(launchers,autoKillTimeout)

1m

Specifies how long to allow a launcher job to run before assuming it is in an error state and killing it.

set CONFIG(tasker,verbose)

1

Controls the logging verbosity of taskers. Range is from 1 (low) to 4 (highly verbose).

set CONFIG(tasker,timeout)

120

Specifies how long the server should give a tasker to establish a connection to the server once its job starts running in the base queue.

set CONFIG(taskerLog,path)

""

Overrides the default tasker log path (`SWD/logs/taskers`).
Logs are organized into a sub-directory structure of `<user>/<baseQueue>/<YYYY.MM.DD>/<HH>`.
If the `<HH>` directory grows to contain more than 5000 files, a new `<HH>.1` directory is created, then a `<HH>.2`, and so on.

set CONFIG(taskerLog,older)

1d

Remove tasker logs older than this setting.
Taskers do not update log files unless there is a need, so long-running taskers may exceed this value. The `vovwxd` daemon keeps this in mind and does not allow removal of a log file for a tasker that is still running.

set CONFIG(delLogDirs,older)

30d

Remove empty tasker/launcher log directories that have been empty for this setting.

set CONFIG(client,derate)

50

Used in the calculation of the maximum number of taskers to allow:

- <number of currently available file descriptors for vovserver> - 2x CONFIG(client,derate)

set CONFIG(launchers,dirname)

"./launchers"
Specifies the path of the launchers directory, the default is relative to `SWD/vovwxd`.
Place into `/tmp`, for example, to avoid launcher performance due to slow NFS performance.

set CONFIG(launchers,checkfreq)

1d
How often to check the launcher and tasker logs directories for cleanup of empty directories, based on the delLogDirs,olderparameter.

set CONFIG(launchers,maxLaunchersPerBucketPerCycle)

99
Specifies the maximum number of launcher jobs that can be submitted per `vovwxd` cycle.

set CONFIG(launchers,quota)

10
Specifies the percentage of launcher jobs initially allowed to be submitted for a new bucket.
This is a method of measuring the base queue responsiveness and throttling tasker requests accordingly.
For example, if a bucket requires 1000 taskers (it has 1000 queued jobs), `vovwxd` will first ask for 10% (the quota value) of the taskers needed. If the base queue responds quickly, all 100 taskers can immediately start running and serving the jobs, then `vovwxd` will double the percentage on the next cycle, otherwise it'll keep requesting 10%.

set CONFIG(savePendingRunningReport)

0
Enables/disables generation of a CSV report file upon each `vovwxd` cycle. Enabled=1;

set CONFIG(queueName,ssl)

0
Specifies whether the base queue matching queueName has SSL enabled for its web interface. Enabled=1.

set CONFIG(failedAgentsCooldownPeriod)

0
Specifies a period in vov time format that vovwxd should wait before requesting taskers from a non responsive base queue.
If this parameter is "0", vovwxd dequeues all jobs in the buckets processed for the failed queue.

Using Direct Drive with vovwxd

Direct Drive provides a method to reduce latency of job start times and improved scaling as more base queues are added.

Accelerator Plus and FlowTracer utilize a Direct Drive mechanism in `vovwxd` to bypass the driver script and speed up tasker requests into the base queue. The WXLauncher tasker is no longer required and the use of launcher jobs is avoided. The call backs defined in the `vovnc.tcl` script are no longer active, although a future version may reinstate some of them.

> **Note:** Direct Drive only works using an NC base queue 2021.1.1 or higher.

1. Set up a WX queue with one or more base NC queues.
2. Copy the `vovaccel.tcl` file to the `SWD/vovwxd` directory.
3. In the `vovwxd config.tcl` file, set the following parameters:

```
set CONFIG(plugin)    "libvovaccel.so"
set CONFIG(driver_script)      "vovaccel.tcl"
```

When a request is issued, `vovwxd` will create a copy of the local bucket in the base queue. This bucket will appear as "foreign bucket" in the base queue. Bucket objects now have new fields to accommodate the foreign buckets: `isforeign`, `wxbucketid`, and `wxqueue`. These can be queried with `vovselect`.

> **Note:** DirectDrive uses a different format of a driver script. The CONFIG(driver_script) parameter must specify an appropriate script. An example is available in `$VOVDIR/etc/config/vovwxd/vovaccel.tcl`.
>
> ```
> set CONFIG(driver_script) "vovaccel.tcl"
> ```
>
> Currently only two procedures are supported: OnProcessBucket BUCKETINFO (for filtering buckets) and OnConfigLoaded(CONFIG), which is called when the config file is modified.

(Deprecated) Configure FlowTracer to Interface With LSF

You can configure FlowTracer to allocate jobs on CPUs managed by LSF while avoiding the taxing per-job scheduling overhead imposed by LSF. You do this by adding the daemon program `vovlsfd` to the set of service programs that run in the background.

The model is that jobs that request an LSF queue resource containing the prefix "LSFqueue:" are candidates to be dispatched by program `vovlsfd`. Additional LSF resources containing the prefix "LSFresource" can be specified.

On demand, `vovlsfd` submits to LSF a request to execute `vovtaskerroot`. Once `vovtaskerroot` connects, FlowTracer can use it to execute one or more jobs that request LSFqueue resources (and optional LSFresources).

Configuring the vovlsf Wrapper

`vovlsfd` makes use of the `vovlsf` wrapper script that is included as part of the installation. The configuration file for `vovlsf` is located at `$VOVDIR/local/vovlsf.config.csh`. If the file does not exist, create it and populate it with information to configure a shell to use your site's particular LSF setup. Once setup, the following commands should work in a `vovproject enabled` session.

```
% vovlsf bsub sleep 10
Job <37655> is submitted to default queue <normal>.
% vovlsf bjobs
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
37655   jeremym PEND  normal     buffalo                 sleep 10   Mar  6 11:56
% vovlsf bqueues
```

```
QUEUE_NAME      PRIO STATUS          MAX JL/U JL/P JL/H NJOBS  PEND   RUN  SUSP
normal           30   Open:Active      -    -    -    -     1      0     1     0
% vovlsf bkill 37655
Job <37655> is being terminated
```

Configure vovlsfd

In order to use `vovlsfd`, the configuration file `PROJECT.swd/vovlsfd/config.tcl` must exist.

You can start from the example shown here, by copying it from `$VOVDIR/etc/config/vovlsfd/config.tcl`.

```
# Example Configuration file for RTDA vovlsfd:
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Remember to add the resource "LSFqueue:$myqueue" to your LSF jobs,
# or vovlsfd will assume the jobs are for direct/already connected taskers.

# How often should the vovlsfd daemon cycle?
# The value is a VOV time spec and the default is two seconds.
set VOVLSFD(refresh)                      2s

# how frequently should we ask bjobs for status?
# The value is a VOV time spec and the default value is one minute.
set VOVLSFD(bjobs,checkfreq)              1m

# How often should we check for sick taskers?
# The value is a VOV time spec and the default is one minute
set VOVLSFD(sick,checkfreq)               1m

# Remove sick taskers that are older than?
# Value is a VOV time spec, and the default value is five minutes.
set VOVLSFD(sick,older)                   5m

# Should we dequeue any extra taskers?
# Setting this to "1" will cause a dequeue
# of all not yet running vovtasker submissions for a job bucket.
# This only happens after three consecutive refresh cycles
# have gone by with no work scheduled for that bucket.
set VOVLSFD(dequeueExtraTaskersEnable) 1

# How long should we wait to dequeue any extra taskers?
# The number of refresh cycles
set VOVLSFD(dequeueExtraTaskersDelay)  3

# What is the maximum number taskers we should start?
# Should be set to a high value to enable lots of parallelism.
set VOVLSFD(tasker,max)                    99

# What is the maximum number of queued taskers per bucket that we should allow?
set VOVLSFD(tasker,maxQueuedPerBucket) 99

# How many tasker submissions should be done for each
# job bucket, during each refresh cycle?
set VOVLSFD(tasker,maxSubsPerBucket)    1

# What is the minimum number of taskers that will be grouped into an array
# for each resource bucket, during each refresh cycle?
# Setting this to "0" will disable LSF array functionality.
set VOVLSFD(tasker,lsfArrayMin)    0

# What is the maximum number of taskers that will be grouped into an array
# for each resource bucket, during each refresh cycle? The absolute max number
# of taskers supercedes this value.
# Setting this to "0" will disable LSF array functionality.
set VOVLSFD(tasker,lsfArrayMax)    0

# What is the longest a vovtasker should run before self-exiting?
# Ex: if you set it to 8 hours, and queue 4 3-hour jobs:
# the first tasker will run for nine hours (3 x 3-hr > 8-hr) and then exit
# the fourth job will only start when a second tasker has been requeued
# and started by the batch execution system.
# This controls the amount of reuse of a tasker while it processes jobs.
# To avoid the penalties of:
# noticing a tasker is needed
# + submitting to the batch system
# + the batch system to allocate a machine
# You should set this to a high value like a week.
# THe value is a VOV time spec
# This is a default value. It can be overriden on a per job basis by putting
# a resource on the job that looks similar to the following.
# MAXlife:1w
set VOVLSFD(tasker,maxlife)        1w

# How long should a tasker wait idle for a job to arrive?
# The shorter time, the faster the slot is released to the batch system.
# The longer time, the more chances the tasker will be reused.
# The default value is two minutes (usually takes a minute to allocate a
# slot through a batch system).  Value is a VOV time spec
# This is a default value. It can be overriden on a per job basis by putting
# a resource on the job that looks similar to the following.
# MAXidle:2m
set VOVLSFD(tasker,maxidle)        2m

# Are there any extra resources you wish to pass along to the taskers?
# These resources will be passed directly along to the vovtasker. They
# are not processed in any way by vovlsfd. For example setting
# this to "MAXlife:1w" will not work as you might expect.
set VOVLSFD(tasker,res)            ""

# Is there a default string you wish to pass as "-P" on the bsub
# command line. Per job settings are also supported by putting a
# resource on the job that looks similar to the following.
# LSFptag:blah
set VOVLSFD(tasker,ptag)           ""

# What is the vovtasker update interval for resource calculation?
# Value is a VOV time spec, and the default value is 15 seconds.
set VOVLSFD(tasker,update)         15s

# How many MB of RAM should we request by default?
# (Default: 256MB)
set VOVLSFD(tasker,ram)            256

# How many CPUs on the same machine should we request by default?
# (Default: 1 core)
set VOVLSFD(tasker,cpus)           1

# What is the default LSF jobname to be used when launching vovtaskers?
set VOVLSFD(tasker,jobname)        "ft_$env(VOV_PROJECT_NAME)"

# Do we want to enable debug messages in the vovtasker log files?
# 0=no; 1=yes; default=0
set VOVLSFD(tasker,debug)          0

# What level of verbosity should the vovtasker use when writing to its the log file?
# Valid values are 0-4; default=1
set VOVLSFD(tasker,verbose)       1

# How long should the vovtasker try to establish the initial connection to
 the vovserver?
# Values are in seconds, default is 120 seconds.
set VOVLSFD(tasker,timeout)       120

# How many hosts should the vovtasker span?
# The value can be overriden by an individual job using the subordinate resource
 functionality.
# Without that override, this config variable defines a default value for
 span[hosts=*]
# Setting the value to zero omits the default span[hosts=*] and allows the underlying
 batch
# scheduler to use its own internally set value.
# Setting to -1 disables the span setting defined for the queue in the batch
 scheduler configuration.
# Setting to 1 tells the batch system to provide all processors from a single host.
# https://www.ibm.com/support/knowledgecenter/SSETD4_9.1.3/lsf_admin/span_string.html
# Valid values are -1, 0, and 1; default=1
set VOVLSFD(tasker,spanHosts)     1

# Which precmd script should we include in the bsub command line by default?
# The precmd script must be an executable script located in the
 VOVLSFD(launchers.dirname) directory.
# The script should be self contained so that we can reference it with a single word.
 (i.e. no arguments, and not full paths)
set VOVLSFD(tasker,precmd)        ""

# How much buffer should we consider when adjusting tasker,max based on available
 client connections?
# This number will be subtracted from the available maxNormalClient connections
# Twice this number will be subtracted from the available file descriptors
# Set this number based on how many non-vovtasker client connections are anticipated
 for this session.
set VOVLSFD(client,derate)       50

# What is the name of the launchers directory? (Default: \"./launchers\")
set VOVLSFD(launchers,dirname)   "./launchers"

# How often should we check the launchers directory for a cleanup?
# The value is a VOV time spec and the default is 10 minutes
set VOVLSFD(launchers,checkfreq) 10m

# Remove launchers that are older than?
# Value is a VOV time spec, and the default value is one hour.
set VOVLSFD(launchers,older)     1h

# Generate a CSV report file (pendingrunning.csv) upon each pass
# of the daemon cycle.
set VOVLSFD(savePendingRunningReport) 0

# queue-specific options:
# set VOVLSFD(bsub,options,$qName)

## To control the name of the executable that is dispatched to LSF.
## The -sr flag in vov_lsf_agent is what
# set VOVLSFD(exeleaf) "$env(VOVDIR)/scripts/vovboot vov_lsf_agent"
# set VOVLSFD(exeleafargs) "-sr"
```

This file does not initially exist, so it will have to be manually created. Use the above example as a template. Here is a sequence of commands to set up `vovlsfd` for a given project.

```
% vovproject enable <project>
% cd `vovserverdir -p .`
% mkdir vovlsfd
% cp $VOVDIR/etc/config/vovlsfd/config.tcl vovlsfd/config.tcl
% vi vovlsfd/config.tcl ; # Edit config file to suit your installation.
```

Start vovlsfd Manually

Use `vovdaemonmgr` to start `vovlsfd` manually.

```
% vovproject enable <project>
% vovdaemonmgr start vovlsfd
```

Start vovlsfd Automatically

In the directory `vnc.swd/autostart` create a script called `vovlsfd.csh` with the following content:

```
#!/bin/csh -f

vovdaemonmgr start vovlsfd
```

Don't forget to make the script executable.

```
% chmod 755 vovlsfd.csh
```

Debug vovlsfd on the Command Line

When first starting `vovlsfd`, it is helpful to run it in foreground, possibly with the -v and/or -d options to verify operation as expected.

```
% vovproject enable <project>
% cd `vovserverdir -p vovlsfd`
% vovlsfd  ; # Launch vovlsfd
```

Specifying LSF Job Resources

To submit a job that is sent to be executed on a LSF CPU, assign the job the following resources:

Resource Name	**Explanation**	**Notes**
LSFqueue:<QUEUENAME>	Specify the LSF queue for the job.	required
LSFresource:<LICENSENAME>	Specify the LSF resource for the job.	optional
LSFlicense:<LSF_LICENSE>	This maps to a "rusage" spec of the type `$LSF_LICENSE=1:duration=1`	optional

Resource Name	Explanation	Notes
LSFapp:<LSF_APP>	This maps to `-app $LSF_APP`	optional
LSFmopts:<LSF_MOPT>	This maps to `-m $LSF_MOPT`	optional
LSFjobname:<LSF_JOBNAME>	This specifies the name of the job in LSF	optional
LSFptag:<LSF_PTAG>	This maps to `-p $LSF_PTAG`	optional
Type:<LSF_TYPE>	This maps into a "select" statement of the form type= $LSF_TYPE	optional
RAM/<ram>	This maps into a "rusage" statement of the form mem= $ram	optional
RAMFREE/<ram>	This maps into a "rusage" statement of the form mem= $ram	optional
CPUS/<cpus>	This maps to `-n $cpus`	optional
VIEW:<VIEWNAME>	Specify the ClearCase (a registered trademark of IBM) view to be used for the job. This resource is automatically added if you use the `-clearcaseView` option of `nc run` and you are in a ClearCase view (i.e. CLEARCASE_ROOT is set).	optional
MAXlife:<TIME>	This overrides the default maxlife value given to the launched vovtasker. `$TIME` is a VOV timespec.	optional
MAXidle:<TIME>	This overrides the default maxidle value given to the launched vovtasker. `$TIME` is a VOV timespec.	optional
vovlsfd:<RESOURCE_NAME>	By declaring a resource token `vovlsfd:$RESOURCE_NAME` in `resources.tcl` with some finite limit, and using the resource on a job, `vovlsfd` will track	optional

Resource Name	Explanation	Notes
	the resource so as not to over submit jobs to LSF.	
TAG:<TAG>	This allows attributes to be passed to the resulting vovtasker, without being considered by the batch system. This allows the user to force jobs to execute only on vovtaskers that have the corresponding <TAG>.	optional

Submitting Jobs to LSF using FlowTracer (FDL)

Examples of assigning LSF resources to jobs with FlowTracer:

```
N "spice"
R "LSFqueue:normal"
J vw spice abc.spi

N "dc_shell"
R "LSFqueue:night LSFresource:dc"
J vw dc_shell -f script.tcl
```

It is also possible to pass additional LSF bsub options for a particular job to `vovlsfd` in two ways.

- Applying the **"BSUB_OPTIONS"** text property to a job
- Adding a specially formatted string (beginning with "--") to the end of the FDL 'R' procedure argument

It's up to the user to determine which approach to use. It's a matter of preference.

```
# 1) Append subordinate string to resource string using the following format :
# R {<required resources> -- "<LSF bsub options>"}
N "md5"
R {LSFqueue:medium -- "-R 'select[type==64BIT]' -app 'highpri'"}
J vw md5.pl

# 2) or, apply a property to the job
N "md5"
R "LSFqueue:medium"
set jobId [J vw md5.pl]
vtk_prop_set -text $jobId BSUB_OPTIONS {-R 'select[type==64BIT]' -app 'highpri'}
```

Please make note that anything in Tcl square brackets will be interpreted as a Tcl command when contained within a double-quoted string, so to prevent any Tcl mischief, either square brackets must be back-slashed, or use curly brackets in place of double quotes.

Control the Values of vovlsfd Options

Because a normal user can change the `config.tcl` file and or change the resource settings for a job to modify LSF or vovtasker settings, it may be desirable to limit or police certain values to keep them in line with enterprise level policy. This can now be done by writing specially named Tcl procedures, and including those procedures in `$VOVDIR/local/vovlsfd/police.tcl`.

A police proc should be named `police_` concatenated with the array index name of the config variable in question. For example, if you want to write a proc that polices the *`VOVLSFD`*`(`tasker`,`*`maxlife)`* setting used to set the maximum life of a vovtasker submitted to LSF by `vovlsfd`, then the proc name is `police_tasker,maxlife`. The proc will be called with a single input, the current value of the variable. The proc should return the desired value.

```
# Example police.tcl proc that limits maxlife to five minutes or less.
proc police_tasker,maxlife { input } {
    if { [VovParseTimeSpec $input] > 300 } {
        return [vtk_time_pp 300]
    } else {
        return $input
    }
}
```

Note: Currently, only the *`maxlife`* and *`maxidle`* values in the `config.tcl` and the per job resource line are guaranteed to work with this new feature. Other variables can be added given demand and/or time for development.

Interface to LSF with vovlsfd

The daemon `vovlsfd` enables Accelerator or FlowTracer to allocate jobs on CPUs managed by LSF while avoiding the taxing per-job scheduling overhead imposed by LSF.

Jobs that request an LSF queue resource containing the prefix "LSFqueue:" are candidates to be dispatched by `vovlsfd`.

The basic idea is that, on demand, `vovlsfd` submits to LSF a request to execute `vovtaskerroot`. Once `vovtaskerroot` connects, Accelerator or FlowTracer can use it to execute one or more jobs that request a LSFqueue resource.

Configure the vovlsf Wrapper

`vovlsfd` makes use of the `vovlsfd` wrapper script that is included as part of the installation. The configuration file for `vovlsfd` is located in `$VOVDIR/local/vovlsf.config.csh`. If the file does not exist, create it and populate it with information to configure a shell to use your site's particular LSF setup. Once setup, the following commands should work in with a `vovproject` enabled terminal.

```
% vovlsf bsub sleep 10
Job <37655> is submitted to default queue <normal>.
% vovlsf bjobs
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
37655   jeremym PEND  normal     buffalo                 sleep 10   Mar  6 11:56
% vovlsf bqueues
QUEUE_NAME      PRIO STATUS          MAX JL/U JL/P JL/H NJOBS  PEND   RUN  SUSP
normal           30  Open:Active       -    -    -    -     1     0     1     0
% vovlsf bkill 37655
Job <37655> is being terminated
```

Configure vovlsfd

In order to use `vovlsfd`, the configuration file `PROJECT.swd/vovlsfd/config.tcl` must exist.

You can start from the example shown here, by copying it from `$VOVDIR/etc/config/vovlsfd/config.tcl`.

```
# Example Configuration file for vovlsfd:
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Remember to add the resource "LSFqueue:$myqueue" to your LSF jobs,
# or vovlsfd will assume the jobs are for direct/already connected taskers.

# How often should the vovlsfd daemon cycle?
# The value is a VOV time spec and the default is two seconds.
set VOVLSFD(refresh)                   2s

# how frequently should we ask bjobs for status?
# The value is a VOV time spec and the default value is one minute.
set VOVLSFD(bjobs,checkfreq)           1m

# How often should we check for sick taskers?
# The value is a VOV time spec and the default is one minute
set VOVLSFD(sick,checkfreq)            1m

# Remove sick taskers that are older than?
# Value is a VOV time spec, and the default value is five minutes.
set VOVLSFD(sick,older)                5m

# Should we dequeue any extra taskers?
# Setting this to "1" will cause a dequeue
# of all not yet running vovtasker submissions for a job bucket.
# This only happens after three consecutive refresh cycles
# have gone by with no work scheduled for that bucket.
set VOVLSFD(dequeueExtraTaskersEnable) 1

# How long should we wait to dequeue any extra taskers?
# The number of refresh cycles
set VOVLSFD(dequeueExtraTaskersDelay)  3

# What is the maximum number taskers we should start?
# Should be set to a high value to enable lots of parallelism.
set VOVLSFD(tasker,max)                 99

# What is the maximum number of queued taskers per bucket that we should allow?
set VOVLSFD(tasker,maxQueuedPerBucket) 99

# How many tasker submissions should be done for each
# job bucket, during each refresh cycle?
set VOVLSFD(tasker,maxSubsPerBucket)   1

# What is the minimum number of taskers that will be grouped into an array
# for each resource bucket, during each refresh cycle?
# Setting this to "0" will disable LSF array functionality.
set VOVLSFD(tasker,lsfArrayMin)    0

# What is the maximum number of taskers that will be grouped into an array
# for each resource bucket, during each refresh cycle? The absolute max number
# of taskers supercedes this value.
# Setting this to "0" will disable LSF array functionality.
set VOVLSFD(tasker,lsfArrayMax)    0

# What is the longest a vovtasker should run before self-exiting?
# Ex: if you set it to 8 hours, and queue 4 3-hour jobs:
```

```
# the first tasker will run for nine hours (3 x 3-hr > 8-hr) and then exit
# the fourth job will only start when a second tasker has been requeued
# and started by the batch execution system.
# This controls the amount of reuse of a tasker while it processes jobs.
# To avoid the penalties of:
# noticing a tasker is needed
# + submitting to the batch system
# + the batch system to allocate a machine
# You should set this to a high value like a week.
# THe value is a VOV time spec
# This is a default value. It can be overriden on a per job basis by putting
# a resource on the job that looks similar to the following.
# MAXlife:1w
set VOVLSFD(tasker,maxlife)        1w

# How long should a tasker wait idle for a job to arrive?
# The shorter time, the faster the slot is released to the batch system.
# The longer time, the more chances the tasker will be reused.
# The default value is two minutes (usually takes a minute to allocate a
# slot through a batch system).  Value is a VOV time spec
# This is a default value. It can be overriden on a per job basis by putting
# a resource on the job that looks similar to the following.
# MAXidle:2m
set VOVLSFD(tasker,maxidle)        2m

# Are there any extra resources you wish to pass along to the taskers?
# These resources will be passed directly along to the vovtasker. They
# are not processed in any way by vovlsfd. For example setting
# this to "MAXlife:1w" will not work as you might expect.
set VOVLSFD(tasker,res)            ""

# Is there a default string you wish to pass as "-P" on the bsub
# command line. Per job settings are also supported by putting a
# resource on the job that looks similar to the following.
# LSFptag:blah
set VOVLSFD(tasker,ptag)           ""

# What is the vovtasker update interval for resource calculation?
# Value is a VOV time spec, and the default value is 15 seconds.
set VOVLSFD(tasker,update)         15s

# How many MB of RAM should we request by default?
#  (Default: 256MB)
set VOVLSFD(tasker,ram)            256

# How many CPUs on the same machine should we request by default?
#  (Default: 1 core)
set VOVLSFD(tasker,cpus)           1

# What is the default LSF jobname to be used when launching vovtaskers?
set VOVLSFD(tasker,jobname)        "ft_$env(VOV_PROJECT_NAME)"

# Do we want to enable debug messages in the vovtasker log files?
# 0=no; 1=yes; default=0
set VOVLSFD(tasker,debug)          0

# What level of verbosity should the vovtasker use when writing to its the log file?
# Valid values are 0-4; default=1
set VOVLSFD(tasker,verbose)        1

    # How long should the vovtasker try to establish the initial connection to the
 vovserver?
# Values are in seconds, default is 120 seconds.
```

```
set VOVLSFD(tasker,timeout)       120

    # How many hosts should the vovtasker span?
# The value can be overriden by an individual job using the subordinate resource
 functionality.
# Without that override, this config variable defines a default value for
 span[hosts=*]
# Setting the value to zero omits the default span[hosts=*] and allows the underlying
 batch
# scheduler to use its own internally set value.
# Setting to -1 disables the span setting defined for the queue in the batch
 scheduler configuration.
# Setting to 1 tells the batch system to provide all processors from a single host.
# https://www.ibm.com/support/knowledgecenter/SSETD4_9.1.3/lsf_admin/span_string.html
# Valid values are -1, 0, and 1; default=1
set VOVLSFD(tasker,spanHosts)     1

# Which precmd script should we include in the bsub command line by default?
# The precmd script must be an executable script located in the
 VOVLSFD(launchers.dirname) directory.
# The script should be self contained so that we can reference it with a single word.
 (i.e. no arguments, and not full paths)
set VOVLSFD(tasker,precmd)        ""

# How much buffer should we consider when adjusting tasker,max based on available
 client connections?
# This number will be subtracted from the available maxNormalClient connections
# Twice this number will be subtracted from the available file descriptors
# Set this number based on how many non-vovtasker client connections are anticipated
 for this session.
set VOVLSFD(client,derate)       50

# What is the name of the launchers directory? (Default: \"./launchers\")
set VOVLSFD(launchers,dirname)   "./launchers"

# How often should we check the launchers directory for a cleanup?
# The value is a VOV time spec and the default is 10 minutes
set VOVLSFD(launchers,checkfreq) 10m

# Remove launchers that are older than?
# Value is a VOV time spec, and the default value is one hour.
set VOVLSFD(launchers,older)     1h

# Generate a CSV report file (pendingrunning.csv) upon each pass
# of the daemon cycle.
set VOVLSFD(savePendingRunningReport) 0

# queue-specific options:
# set VOVLSFD(bsub,options,$qName)

## To control the name of the executable that is dispatched to LSF.
## The -sr flag in vov_lsf_agent is what
# set VOVLSFD(exeleaf) "$env(VOVDIR)/scripts/vovboot vov_lsf_agent"
# set VOVLSFD(exeleafargs) "-sr"
```

This file does not initially exist, so it will have to be manually created. Use the above example as a template. Here is a sequence of commands to set up `vovlsfd` for a given project.

```
% vovproject enable <project>
% cd `vovserverdir -p .`
% mkdir vovlsfd
% cp $VOVDIR/etc/config/vovlsfd/config.tcl vovlsfd/config.tcl
% vi vovlsfd/config.tcl ; # Edit config file to suit your installation.
```

Start vovlsfd Manually

Use `vovdaemonmgr` to start `vovlsfd` manually.

```
% vovproject enable <project>
% vovdaemonmgr start vovlsfd
```

Start vovlsfd Automatically

In the directory `vnc.swd/autostart`, create a script called `vovlsfd.csh` with the following content:

```
#!/bin/csh -f

vovdaemonmgr start vovlsfd
```

Don't forget to make the script executable.

```
% chmod 755 vovlsfd.csh
```

Start Multiple vovlsfd Daemons For Each User

While a single `vovlsfd` daemon is sufficient even when dealing with multiple users, it may be useful at times to have multiple `vovlsfd` daemons running for each user when running FlowTracer in a multi-user operating mode, mostly because it allows traceability of the user in LSF.

To enable the FlowTracer multi-user mode, set the environment variable VOV_FT_MULTIUSER. When `vovlsfd` is started with this environment variable set, each user can start his/her own instance of `vovlsfd`. And this instance will only process jobs for its owner user.

It is the responsibility of the user to keep his own `vovlsfd` daemon running properly. Each private `vovlsfd` daemon uses the config file from its working directory, which is under `<PROJECT>.swd/vovlsfd/<user>`. If the config file is not found in the user specific directory, `vovlsfd` will also check the directory above `<PROJECT>.swd/vovlsfd` for the config file.

Debug vovlsfd on the Command Line

When first starting `vovlsfd`, it is helpful to run it in foreground, possibly with the -v and/or -d options to verify operation as expected.

```
% vovproject enable <project>
% cd `vovserverdir -p vovlsfd`
% vovlsfd  ; # Launch vovlsfd
```

Specify LSF Job Resources

To submit a job that is sent to be executed on a LSF CPU, assign the job the following resources:

Resource Name	Explanation	Notes
LSFqueue:<QUEUENAME>	This maps to `-q <QUEUENAME>`	required

Resource Name	Explanation	Notes
LSFlicense:<LSF_LICENSE>	This maps to a "rusage" spec of the type `<LSF_LICENSE>=1:duration=1`	optional
LSFapp:<LSF_APP>	This maps to `-app <LSF_APP>`	optional
LSFmopts:<LSF_MOPT>	This maps to `-m <LSF_MOPT>`	optional
LSFjobname:<LSF_JOBNAME>	This maps to `-J <LSF_JOBNAME>`	optional
LSFptag:<LSF_PTAG>	This maps to `-p <LSF_PTAG>`	optional
LSFpre:<LSF_PRECMD>	This maps to `-E <LSF_PRECMD>`. Its important that the value provided for <LSF_PRECMD> is a single word with no spaces or slashes. In other words no full paths or arguments. The script itself must be executable and exist in the `launchers/prescripts` directory. Typically, `SWD/vovlsfd/launchers/prescripts`. The LSF:pre special resource will use this directory as the base directory for specified prescript. Example: `LSFpre:mypre.sh` results in an LSF submission option of `-E ./prescripts/mypre.sh`.	optional
Type:<LSF_TYPE>	This maps into a "select" statement of the form type=<LSF_TYPE>	optional
RAM/<ram>	This maps into a "rusage" statement of the form mem=<ram>	optional
CORES/<cpus>	This maps to `-n <cpus>`	optional
MAXlife:<TIMESPEC>	This overrides the default maxlife value given to the launched vovtasker. `<TIMESPEC>` is a VOV timespec.	optional

Resource Name	Explanation	Notes
MAXidle:<TIMESPEC>	This overrides the default maxidle value given to the launched vovtasker. `<TIMESPEC>` is a VOV timespec.	optional
vovlsfd:<RESOURCE_NAME>	By declaring a resource token `vovlsfd:<RESOURCE_NAME>` in `resources.tcl` with some finite limit, and using the resource on a job, `vovlsfd` will track the resource so as not to over submit jobs to LSF.	optional
TAG:<TAG>	This allows attributes to be passed to the resulting vovtasker, without being considered by the batch system. This allows the user to force jobs to execute only on vovtaskers that have the corresponding <TAG>.	optional

Submit Jobs to LSF using Accelerator

Examples of job submission with Accelerator:

```
% nc run -r LSFqueue:normal -- spice abc.spi
% nc run -r LSFqueue:night LSFresource:dc -- dc_shell -f script.tcl
```

Submit Jobs to LSF using FlowTracer (FDL)

Examples of assigning LSF resources to jobs with FlowTracer:

```
N "spice"
R "LSFqueue:normal"
J vw spice abc.spi

N "dc_shell"
R "LSFqueue:night LSFresource:dc"
J vw dc_shell -f script.tcl
```

It is also possible to pass additional LSF bsub options for a particular job to `vovlsfd` in two ways.

1. Applying the "BSUB_OPTIONS" text property to a job
2. Adding a specially formatted string (beginning with "--") to the end of the FDL 'R' procedure argument

It's up to you to determine which approach to use. It's a matter of preference.

```
# 1) Append subordinate string to resource string using the following format :
# R {<required resources> -- "<LSF bsub options>"}
N "md5"
R {LSFqueue:medium -- "-R 'select[type==64BIT]' -app 'highpri'"}
J vw md5.pl

# 2) or, apply a property to the job
N "md5"
R "LSFqueue:medium"
set jobId [J vw md5.pl]
vtk_prop_set -text $jobId BSUB_OPTIONS {-R 'select[type==64BIT]' -app 'highpri'}
```

Make note that text in Tcl square brackets will be interpreted as a Tcl command when contained within a double-quoted string, so to prevent any Tcl mischief, either square brackets must be back-slashed, or use curly brackets in place of double quotes.

Control the Values of vovlsfd Options

Because a normal user can change the `config.tcl` file and or change the resource settings for a job to modify LSF or vovtasker settings, it may be desirable to limit or police certain values to keep them in line with enterprise level policy. This can now be done by writing specially named Tcl procedures, and including those procedures in `$VOVDIR/local/vovlsfd/police.tcl`.

A police proc should be named `police_` concatenated with the array index name of the config variable in question. For example, if you want to write a proc that polices the VOVLSFD(tasker,maxlife) setting used to set the maximum life of a vovtasker submitted to LSF by `vovlsfd`, then the proc name is `police_tasker,maxlife`. The proc will be called with a single input, the current value of the variable. The proc should return the desired value.

```
# Example police.tcl proc that limits maxlife to five minutes or less.
proc police_tasker,maxlife { input } {
    if { [VovParseTimeSpec $input] > 300 } {
        return [vtk_time_pp 300]
    } else {
        return $input
    }
}
```

Note: Currently, only the *`maxlife`* and *`maxidle`* values in the `config.tcl` and the per job resource line are guaranteed to work with this new feature. Other variables can be added given demand and/or time for development.

Automatic Submission to the LSF Batch Queuing System

In large enterprises, there is often a configuration with shared login machines where no CPU-intense process should run on the login machines due to its shared nature. Usually the vovserver and the vovconsole are of low enough CPU usage that they may run on those shared machines. However in some environments the policy may be that no process at all apart from the approved processes like xterm may run on those shared machines. The following is available to dispatch the vovconsole and the vovserver to a remote machine found by the batch system LSF.

Submit the vovconsole

To enable automatic submission of the vovconsole to the LSF batch queuing system, use:

Variable Name	Description
VOVCONSOLE_SUBMIT_CMD	Define the environment variable so that the vovconsole automatically dispatches itself. Typical usage: `setenv VOVCONSOLE_SUBMIT_CMD 'bsub -o /dev/null -R "rusage[mem=200]" '` Use the following to check the value of that variable, as `echo $VOVCONSOLE_SUBMIT_CMD` would return a No match error: `printenv | grep VOVCONSOLE_SUBMIT_CMD` `VOVCONSOLE_SUBMIT_CMD=bsub -R "rusage[mem=200]"` You may also want to use the following to debug submission errors. `setenv VOVCONSOLE_SUBMIT_CMD 'bsub -R "rusage[mem=200]" -I '`

Submitting the vovserver

To enable automatic submission of the vovserver to the LSF batch queuing system, use:

Variable Name	Description
VOVPROJECT_SUBMIT_CMD	This variable contains the command used by 'vovproject' to launch the 'vovserver', for example using bsub. Typical usage: `setenv VOVPROJECT_SUBMIT_CMD 'bsub -o logs/server.20130531_123000.log -R "rusage[mem=200]" '` Since this variable often has a value with tricky characters, we suggest you use : `% printenv | grep VOVPROJECT_SUBMIT_CMD` `VOVPROJECT_SUBMIT_CMD=bsub -R "rusage[mem=200]"` to check the value of that variable as echo $VOVPROJECT_SUBMIT_CMD would return a No match error. For debugging purposes, you may also want to use interactive options like the option -I in this example:

Variable Name	**Description**
	% setenv VOVPROJECT_SUBMIT_CMD 'bsub -R "rusage[mem=200]" -I '

Configuration for Altair Accelerator

Configure Altair FlowTracer to Interface with Altair Accelerator

You can configure FlowTracer to allocate jobs on CPUs managed by LSF while avoiding the per-job scheduling overhead imposed by LSF. You do this by adding the daemon program `vovelasticd` to the set of service programs that run in the background.

The model is that jobs that request an LSF queue resource containing the prefix "LSFqueue:" are ignored by `vovelasticd`. This allows `vovlsfd` and `vovelasticd` to cooperate and work together. All jobs that do not request an LSF queue are candidates for `vovelasticd`.

On demand, `vovelasticd` submits to Accelerator a request to execute `vov_elastic_agent`. Once `vov_elastic_agent` connects, FlowTracer can use it to execute one or more jobs.

In addition, the vovtasker connection timeout for elastic vovtaskers can be configured via the `config.tcl` file.

Configure vovelasticd

In order to use `vovelasticd`, the configuration file `PROJECT.swd/vovelasticd/config.tcl` must exist.

You can start from the example shown here, by copying it from `$VOVDIR/etc/config/vovelasticd/config.tcl`.

```
# How often should the vovelasticd daemon cycle?
# The value is a VOV time spec and the default is two seconds.
set VOVELASTICD(refresh)                    2s

# Which Accelerator instance should the vovelasticd daemon submit jobs to?
set VOVELASTICD(queue)                      "vnc"

# Environment to be used when submitting an vov_elastic_agent.
set VOVELASTICD(runenv)                     "BASE"

# How old should a job bucket be before vovelasticd considers it for servicing?
# The value is a VOV time spec and the default is 5 seconds.
set VOVELASTICD(bucketAgeThreshold)         0

# What should we call jobs that are submitted to Accelerator?
set VOVELASTICD(elasticJobName)             "$daemonName"

# To which set in the Accelerator instance should our jobs be appended?
set VOVELASTICD(elasticSetName)             "$daemonName:$projInfo(name)@
$projInfo(host)@$projInfo(port)"

# How often should we check for failed job submissions?
# When a failed job submission is detected, it is removed
# from the internal data structres used for calculating
# whether or not to submit a new tasker.
set VOVELASTICD(failed,checkfreq)           1m
```

```
# How often should we check for sick taskers?
# The value is a VOV time spec and the default is one minute
set VOVELASTICD(sick,checkfreq)           1m

# Remove sick taskers that are older than?
# Value is a VOV time spec, and the default value is five minutes.
set VOVELASTICD(sick,older)               5m

# Should we dequeue any extra taskers?
# Setting this to "1" will cause a dequeue
# of all not yet running vovtasker submissions for a job bucket.
# This only happens after three consecutive refresh cycles
# have gone by with no work scheduled for that bucket.
set VOVELASTICD(dequeueExtraTaskersEnable) 0

# How long should we wait to dequeue any extra taskers?
# The number of refresh cycles
set VOVELASTICD(dequeueExtraTaskersDelay) 3

# What is the maximum number taskers we should start?
# Should be set to a high value to enable lots of parallelism.
set VOVELASTICD(tasker,max)                99

# What is the maximum number of queued taskers per bucket that we should allow?
set VOVELASTICD(tasker,maxQueuedPerBucket) 99

# How many tasker submissions should be done for each
# resource bucket, during each refresh cycle?
# Setting this to "0" will disable blitz functionality.
set VOVELASTICD(tasker,blitz)              0

# What is the minimum number of taskers that will be grouped into an array
# for each resource bucket, during each refresh cycle?
# Setting this to "0" will disable array functionality.
set VOVELASTICD(tasker,jobArrayMin)        0

# What is the maximum number of taskers that will be grouped into an array
# for each resource bucket, during each refresh cycle? The absolute max number
# of taskers supercedes this value.
# Setting this to "0" will disable array functionality.
set VOVELASTICD(tasker,jobArrayMax)        0

# What is the longest a vovtasker should run before self-exiting?
# Ex: if you set it to 8 hours, and queue 4 3-hour jobs:
# the first tasker will run for nine hours (3 x 3-hr > 8-hr) and then exit
# the fourth job will only start when a second tasker has been requeued
# and started by the batch execution system.
# This controls the amount of reuse of a tasker while it processes jobs.
# To avoid the penalties of:
# noticing a tasker is needed
# + submitting to the batch system
# + the batch system to allocate a machine
# You should set this to a high value like a week.
# THe value is a VOV time spec
# This is a default value. It can be overriden on a per job basis by putting
# a resource on the job that looks similar to the following.
# MAXlife:1w
set VOVELASTICD(tasker,maxlife)            1w

# How long should a tasker wait idle for a job to arrive?
# The shorter time, the faster the slot is released to the batch system.
# The longer time, the more chances the tasker will be reused.
# The default value is two minutes (usually takes a minute to allocate a
# slot through a batch system).  Value is a VOV time spec
# This is a default value. It can be overriden on a per job basis by putting
# a resource on the job that looks similar to the following.
# MAXidle:2m
set VOVELASTICD(tasker,maxidle)             2m

# Are there any extra resources you wish to pass along to the taskers?
# These resources will be passed directly along to the vovtasker. They
# are not processed in any way by vovelasticd. For example setting
# this to "MAXlife:1w" will not work as you might expect.
set VOVELASTICD(tasker,res)                 ""

# What is the vovtasker update interval for resource calculation?
# Value is a VOV time spec, and the default value is 15 seconds.
set VOVELASTICD(tasker,update)              15s

# Do we want to enable debug messages in the vovtasker log files?
# 0=no; 1=yes; default=0
set VOVELASTICD(tasker,debug)               0

# What level of verbosity should the vovtasker use when writing to its the log file?
# Valid values are 0-4; default=1
set VOVELASTICD(tasker,verbose)             1

# How long should the vovtasker try to establish the initial connection to the
 vovserver?
# Values are in seconds, default is 120 seconds.
set VOVELASTICD(tasker,timeout)             120

# How much buffer should we consider when adjusting tasker,max based on available
 client connections?
# This number will be subtracted from the available maxNormalClient connections
# Twice this number will be subtracted from the available file descriptors
# Set this number based on how many non-vovtasker client connections are anticipated
 for this session.
set VOVELASTICD(client,derate)             50

# What is the name of the launchers directory? (Default: \"./launchers\")
set VOVELASTICD(launchers,dirname)         "./launchers"

# How often should we check the launchers directory for a cleanup?
# The value is a VOV time spec and the default is 10 minutes
set VOVELASTICD(launchers,checkfreq)       10m

# Remove launchers that are older than?
# Value is a VOV time spec, and the default value is one hour.
set VOVELASTICD(launchers,older)           1h

# How often should we check for preempted jobs?
# Value is a VOV time spec, and the default value is one minute.
set VOVELASTICD(preempt,checkfreq)         1m

# What action should we take on a prempted vovtasker?
# Allowable values are STOP and RESERVE.
# STOP is faster than RESERVE but requires > 2013.09u5
set VOVELASTICD(preempt,taskerstop)         "STOP"

# Should we allow vovtaskers to be preempted?
# Value is 1 or 0.
# The preemption request must come from an appropriately configured Accelerator.
set VOVELASTICD(preemptTaskersEnable)       0

# In FT 2013.09u6 and possibly later, this setting should be 1,
# if the underlying batch system is Accelerator and NUMA support is required by
 the taskers.
set VOVELASTICD(resources,numamap)        1

# The maximum number of launcher attempts that should be made in the event the job
 submission
# to the back-end queue fails. In most cases, this would be due to a misconfigured
 queue name.
# The counter is reset once the configuration file has been updated.
set VOVELASTICD(maxQueueErrors)           10
```

This file does not initially exist, so it will have to be manually created. Use the above example as a template. Here is a sequence of commands to set up `vovelasticd` for a given project.

```
% vovproject enable <project>
% cd `vovserverdir -p .`
% mkdir vovelasticd
% cp $VOVDIR/etc/config/vovelasticd/config.tcl vovelasticd/config.tcl
% vi vovelasticd/config.tcl ; # Edit config file to suit your installation.
```

Start vovelasticd

Use `vovdaemonmgr` to start `vovelasticd` manually.

```
% vovproject enable <project>
% vovdaemonmgr start vovelasticd
```

To start `vovelasticd` automatically, in the directory `vnc.swd/autostart`, create a script called `vovelasticd.csh` with the following content:

```
#!/bin/csh -f
vovdaemonmgr start vovelasticd
```

Don't forget to make the script executable:

```
% chmod 755 vovelasticd.csh
```

Debug vovelasticd

When first starting `vovelasticd`, it is helpful to run it in foreground, possibly with the -v and/or -d options to verify operation as expected.

```
% vovproject enable <project>
% cd `vovserverdir -p vovelasticd`
% vovelasticd ; # Launch vovelasticd
```

Specify Job Resources

To submit a job that is sent to be executed on an Accelerator CPU, assign the job resources as you normally would for Accelerator. The resource request is passed along to Accelerator without modification. The following resources require special handling by `vovelasticd` internally.

Table 1:

Resource Name	Explanation	Notes
MAXlife:<TIME>	This overrides the default maxlife value given to the launched vovtasker. $TIME is a VOV timespec.	Optional
MAXidle:<TIME>	This overrides the default maxidle value given to the launched vovtasker. $TIME is a VOV timespec.	Optional
vovelasticd:<RESOURCE_NAME>	By declaring a resource token vovelasticd:$RESOURCE_NAME in resources.tcl with some finite limit, and using the resource on a job, vovelasticd will track the resource so as not to over submit jobs to the batch queuing system.	Optional
TAG:<TAG>	This allows attributes to be passed to the resulting vovtasker, without being considered by the batch system. This allows the user to force jobs to execute only on vovtaskers that have the corresponding <TAG>.	Optional

Submit Jobs to Accelerator using FlowTracer (FDL)

Examples of assigning resources to jobs with FlowTracer:

```
N "spice"
R "MAXlife:1h"
J vw spice abc.spi

N "dc_shell"
R "CORES/2 MAXidle:1m"
J vw dc_shell -f script.tcl
```

Interface with Altair Accelerator using vovelasticd

The daemon `vovelasticd` enables Altair Accelerator or Altair FlowTracer to allocate jobs on CPUs managed by Accelerator while avoiding per-job scheduling overhead.

Jobs that request an LSF queue resource containing the prefix "LSFqueue:" are ignored by `vovelasticd`. This allows `vovlsfd` and `vovelasticd` to cooperate and work together. All jobs that do not request an LSF queue are candidates for `vovelasticd`.

The basic idea is that, on demand, `vovelasticd` submits to Accelerator a request to execute `vov_elastic_agent`. Once `vov_elastic_agent` connects, Accelerator or FlowTracer can use it to execute one or more jobs.

In addition, the vovtasker connection timeout for elastic vovtaskers can be configured via the `config.tcl` file.

Configure vovelasticd

In order to use `vovelasticd`, the configuration file `PROJECT.swd/vovelasticd/config.tcl` must exist.

You can start from the example shown here, by copying it from `$VOVDIR/etc/config/vovelasticd/config.tcl`.

```
# How often should the vovelasticd daemon cycle?
# The value is a VOV time spec and the default is two seconds.
set VOVELASTICD(refresh)                    2s

# Which NC instance should the vovelasticd daemon submit jobs to?
set VOVELASTICD(queue)                      "vnc"

# Environment to be used when submitting an vov_elastic_agent.
set VOVELASTICD(runenv)                     "BASE"

# How old should a job bucket be before vovelasticd considers it for servicing?
# The value is a VOV time spec and the default is 5 seconds.
set VOVELASTICD(bucketAgeThreshold)         0

# What should we call jobs that are submitted to NC?
set VOVELASTICD(elasticJobName)             "$daemonName"

# To which set in the NC instance should our jobs be appended?
set VOVELASTICD(elasticSetName)             "$daemonName:$projInfo(name)@
$projInfo(host)@$projInfo(port)"

# How often should we check for failed job submissions?
# When a failed job submission is detected, it is removed
# from the internal data structres used for calculating
# whether or not to submit a new tasker.
set VOVELASTICD(failed,checkfreq)           1m

# How often should we check for sick taskers?
# The value is a VOV time spec and the default is one minute
set VOVELASTICD(sick,checkfreq)             1m

# Remove sick taskers that are older than?
# Value is a VOV time spec, and the default value is five minutes.
set VOVELASTICD(sick,older)                 5m
```

```
# Should we dequeue any extra taskers?
# Setting this to "1" will cause a dequeue
# of all not yet running vovtasker submissions for a job bucket.
# This only happens after three consecutive refresh cycles
# have gone by with no work scheduled for that bucket.
set VOVELASTICD(dequeueExtraTaskersEnable) 0

# How long should we wait to dequeue any extra taskers?
# The number of refresh cycles
set VOVELASTICD(dequeueExtraTaskersDelay) 3

# What is the maximum number taskers we should start?
# Should be set to a high value to enable lots of parallelism.
set VOVELASTICD(tasker,max)                     99

# What is the maximum number of queued taskers per bucket that we should allow?
set VOVELASTICD(tasker,maxQueuedPerBucket) 99

# How many tasker submissions should be done for each
# resource bucket, during each refresh cycle?
# Setting this to "0" will disable blitz functionality.
set VOVELASTICD(tasker,blitz)                   0

# What is the minimum number of taskers that will be grouped into an array
# for each resource bucket, during each refresh cycle?
# Setting this to "0" will disable array functionality.
set VOVELASTICD(tasker,jobArrayMin)             0

# What is the maximum number of taskers that will be grouped into an array
# for each resource bucket, during each refresh cycle? The absolute max number
# of taskers supercedes this value.
# Setting this to "0" will disable array functionality.
set VOVELASTICD(tasker,jobArrayMax)             0

# What is the longest a vovtasker should run before self-exiting?
# Ex: if you set it to 8 hours, and queue 4 3-hour jobs:
# the first tasker will run for nine hours (3 x 3-hr > 8-hr) and then exit
# the fourth job will only start when a second tasker has been requeued
# and started by the batch execution system.
# This controls the amount of reuse of a tasker while it processes jobs.
# To avoid the penalties of:
# noticing a tasker is needed
# + submitting to the batch system
# + the batch system to allocate a machine
# You should set this to a high value like a week.
# The value is a VOV time spec
# This is a default value. It can be overriden on a per job basis by putting
# a resource on the job that looks similar to the following.
# MAXlife:1w
set VOVELASTICD(tasker,maxlife)                 1w

# How long should a tasker wait idle for a job to arrive?
# The shorter time, the faster the slot is released to the batch system.
# The longer time, the more chances the tasker will be reused.
# The default value is two minutes (usually takes a minute to allocate a
# slot through a batch system).  Value is a VOV time spec
# This is a default value. It can be overriden on a per job basis by putting
# a resource on the job that looks similar to the following.
# MAXidle:2m
set VOVELASTICD(tasker,maxidle)                 2m

# Are there any extra resources you wish to pass along to the taskers?
# These resources will be passed directly along to the vovtasker. They
# are not processed in any way by vovelasticd. For example setting
# this to "MAXlife:1w" will not work as you might expect.
set VOVELASTICD(tasker,res)                  ""

# What is the vovtasker update interval for resource calculation?
# Value is a VOV time spec, and the default value is 15 seconds.
set VOVELASTICD(tasker,update)               15s

# Do we want to enable debug messages in the vovtasker log files?
# 0=no; 1=yes; default=0
set VOVELASTICD(tasker,debug)                0

# What level of verbosity should the vovtasker use when writing to its the log file?
# Valid values are 0-4; default=1
set VOVELASTICD(tasker,verbose)              1

# How long should the vovtasker try to establish the initial connection to the
 vovserver?
# Values are in seconds, default is 120 seconds.
set VOVELASTICD(tasker,timeout)              120

# How much buffer should we consider when adjusting tasker,max based on available
 client connections?
# This number will be subtracted from the available maxNormalClient connections
# Twice this number will be subtracted from the available file descriptors
# Set this number based on how many non-tasker client connections are anticipated for
 this session.
set VOVELASTICD(client,derate)              50

# What is the name of the launchers directory? (Default: \"./launchers\")
set VOVELASTICD(launchers,dirname)          "./launchers"

# How often should we check the launchers directory for a cleanup?
# The value is a VOV time spec and the default is 10 minutes
set VOVELASTICD(launchers,checkfreq)        10m

# Remove launchers that are older than?
# Value is a VOV time spec, and the default value is one hour.
set VOVELASTICD(launchers,older)            1h

# How often should we check for preempted jobs?
# Value is a VOV time spec, and the default value is one minute.
set VOVELASTICD(preempt,checkfreq)          1m

# What action should we take on a prempted vovtasker?
# Allowable values are STOP and RESERVE.
# STOP is faster than RESERVE but requires > 2013.09u5
set VOVELASTICD(preempt,taskerstop)          "STOP"

# Should we allow vovtasker to be preempted?
# Value is 1 or 0.
# The preemption request must come from an appropriately configured NC.
set VOVELASTICD(preemptTaskersEnable)        0

# In FT 2013.09u6 and possibly later, this setting should be 1,
# if the underlying batch system is NC and NUMA support is required by the taskers.
set VOVELASTICD(resources,numamap)          1

# The maximum number of launcher attempts that should be made in the event the job
 submission
# to the back-end queue fails. In most cases, this would be due to a misconfigured
 queue name.
# The counter is reset once the configuration file has been updated.
```

```
set VOVELASTICD(maxQueueErrors)            10
```

This file does not initially exist, so it will have to be manually created. Use the above example as a template. Here is a sequence of commands to set up `vovelasticd` for a given project.

```
% vovproject enable <project>
% cd `vovserverdir -p .`
% mkdir vovelasticd
% cp $VOVDIR/etc/config/vovelasticd/config.tcl vovelasticd/config.tcl
% vi vovelasticd/config.tcl ; # Edit config file to suit your installation.
```

Start vovelasticd Manually

Use `vovdaemonmgr` to start `vovelasticd` manually.

```
% vovproject enable <project>
% vovdaemonmgr start vovelasticd
```

Start vovelasticd Automatically

In the directory `vnc.swd/autostart` create a script called `vovelasticd.csh` with the following content:

```
#!/bin/csh -f
vovdaemonmgr start vovelasticd
```

Don't forget to make the script executable.

```
% chmod 755 vovelasticd.csh
```

Debug vovelasticd on the Command Line

When first starting `vovelasticd`, it is helpful to run it in foreground, possibly with the -v and/or -d options to verify operation as expected.

```
% vovproject enable <project>
% cd `vovserverdir -p vovelasticd`
% vovelasticd ; # Launch vovelasticd
```

Specify Job Resources

To submit a job that is sent to be executed on an Accelerator CPU, assign the job resources as you normally would for Accelerator. The resource request is passed along to Accelerator without modification. The following resources require special handling by `vovelasticd` internally.

Resource Name	Explanation	Notes
MAXlife:<TIME>	This overrides the default maxlife value given to the launched vovtasker. `$TIME` is a VOV timespec.	optional
MAXidle:<TIME>	This overrides the default maxidle value given to the	optional

Resource Name	Explanation	Notes
	launched vovtasker. `$TIME` is a VOV timespec.	
vovelasticd:<RESOURCE_NAME>	By declaring a resource token `vovelasticd:$RESOURCE_NAME` in `resources.tcl` with some finite limit, and using the resource on a job, `vovelasticd` will track the resource so as not to over submit jobs to the batch queuing system.	optional
TAG:<TAG>	This allows attributes to be passed to the resulting vovtasker, without being considered by the batch system. This allows the user to force jobs to execute only on vovtaskers that have the corresponding <TAG>.	optional

Submit Jobs to Accelerator using Accelerator

Examples of job submission with Accelerator:

```
% nc run -r MAXlife:1h -- spice abc.spi
% nc run -r CORES/2 MAXidle:1m -- dc_shell -f script.tcl
% nc run -clearcaseResource -r CORES/2 vovelasticd:dcThrottle/2 -- dc_shell -f
 script.tcl
```

Submit Jobs to Accelerator using FlowTracer (FDL)

Examples of assigning resources to jobs with FlowTracer:

```
N "spice"
R "MAXlife:1h"
J vw spice abc.spi

N "dc_shell"
R "CORES/2 MAXidle:1m"
J vw dc_shell -f script.tcl
```

Configure a Failover Server Replacement

If a server crashes suddenly, VOV has the capability to start a replacement server on a pre-selected host. This capability requires that the pre-selected host is configured as a failover server.

The configuration instructions follow.

Note: The `vovserverdir` command only works from a VOV-enabled shell when the project server is running.

1. Edit or create the file `servercandidates.tcl` in the server configuration directory. Use the `vovserverdir` command with the -p option to find the pathname to this file.

   ```
   % vovserverdir -p servercandidates.tcl
   /home/john/vov/myProject.swd/servercandidates.tcl
   ```

   The `servercandidates.tcl` file should set the Tcl variable *ServerCandidates* to a list of possible failover hosts. This list may include the original host on which the server was started.

   ```
   set ServerCandidates {
       host1
       host2
       host3
   }
   ```

2. Install the `autostart/failover.sh` script as follows:

   ```
   % cd `vovserverdir -p .`
   % mkdir autostart
   % cp $VOVDIR/etc/autostart/failover.sh autostart/failover.sh
   % chmod a+x autostart/failover.sh
   ```

3. Activate the failover facility by running `vovautostart`.

   ```
   % vovautostart
   ```

   For example:

   ```
   % vovtaskermgr show -taskergroups
   ID          taskername         hostname          taskergroup
   000404374   localhost-2       titanus          g1
   000404375   localhost-1       titanus          g1
   000404376   localhost-5       titanus          g1
   000404377   localhost-3       titanus          g1
   000404378   localhost-4       titanus          g1
   000404391   failover          titanus          failover
   ```

   Note: Each machine listed as a server candidate must be a vovtasker machine; the vovtasker running on that machine acts as its agent in selecting a new server host. Taskers can be configured as dedicated failover candidates that are not allowed to run jobs by using the `-failover` option in the taskers definition.

Preventing jobs from running on the candidate machine eliminates the risks of machine stability being affected by demanding jobs. The `-failover` option also enables some failover configuration validation checks. Failover taskers are started before the regular queue taskers, which helps ensure a failover tasker is available as soon as possible for future failover events.

The `-failover` option keeps taskers up and running without the need for jobs to be running (in fact, jobs are not allowed to run on them). This allows them to participate in the server election process and start up vovserver without introducing a competition for resources.

Without `-failover`, the taskers will be normal ones, which can run jobs, and in fact, they *must* be running jobs at the time of the vovserver kill/crash because without no jobs, the taskers will exit:

```
if ( ss_activeTransitions <= 0  && (! isInFailoverGroup ) ) {
   addLog( "No jobs running: no need to keep running." );
   goto tasker_exit;
}
```

The easiest way to ensure the tasker is in the failover group is to use the `-failover` option.

Refer to the tasker definition documentation for details on the `-failover` option.

How vovserver Failover Works

If the vovserver crashes, after a period of time, the vovtasker process on each machine notices that it has had no contact from the server, and it initiates a server machine election.

In this election, each vovtasker votes for itself (precisely, the host that this particular tasker runs on) as a server candidate. The election is conducted by running the script `vovservsel.tcl`.

After the time interval during which the vovtaskers vote expires, (default 60 seconds) the host that appears earliest on the list will be selected to start a new vovserver.

In the following example, the `servercandidates.tcl`, file contains three hosts:

```
set ServerCandidates {
    host1
    host2
    host3
}
```

When the server crashes, if there are vovtaskers running on host1, host2 and host3, then these hosts will be voted as server candidates. Then host2 will be the best candidate and a new vovserver will be started on host2. This server will start in crash recovery mode.

Note: For failover recovery to be successful, an active `vovtasker` process must be running on at least one of the hosts named in the `ServerCandidates` list. Usually, these vovtaskers have been defined with the `-failover` option so they can not accept any jobs, and are members of the `failover` taskergroup.

The failover vovserver will read the most-recently-saved PR file from the `.swd/trace.db` directory, and then read in the transactions from the 'cr*' (crash recovery) files to recover as much of the pre-crash state as possible.

The vovserver writes a new `serverinfo.tcl` file in the `.swd` that vovtaskers read to determine the port and host. When it starts, the failover vovserver appends the new host and port information to the `$NC_CONFIG_DIR/<queue-name.tcl>` as well as to the `setup.tcl` in the server configuration directory. The vovserver then runs the scripts in the autostart directory. This should include the `failover.sh` script, which resets the failover directory so that failover can repeat. This script removes the registry entry, and removes the `server_election` directory and creates a new empty one. At the end, it calls `vovproject reread` to force the failover vovserver to create an updated registry entry.

The failover vovserver remains in crash recovery mode for an interval, usually one minute, waiting for any vovtaskers that have running jobs to reconnect:

- For Accelerator, Accelerator Plus, Monitor and Allocator, vovtaskers wait up to 4 days for a new server to start.
- For FlowTracer, vovtaskers wait up to 3 minutes for a new server to start.

After reconnecting to vovserver, vovtaskers automatically exit after all of their running jobs are completed. After the vovserver transitions from crash recovery mode to normal mode, it will try to restart any configured vovtaskers that are not yet running.

Any of the following conditions will prevent successful failover server restart:

- The filesystem holding the `.swd` directory is unavailable.
- The file `servercandidates.tcl` does not exist.
- The `ServerCandidates` list is empty.
- There is no vovtasker running on any host in the `ServerCandidates` list when the server crashes.
- The `autostart/failover.sh` script file is not in place.

In this case, the failover server will not be automatically started; the server will have to be manually started.

Tips for Configuring Failover

Following are tips for failover configuration:

- Make the first failover host the regular one. This way, if the vovserver dumps core or is killed by mistake, it will restart on the regular host.
- Configure special vovtaskers only for failover by passing the `-failover` option to `vtk_tasker_define`.
- Test that failover works before depending on it.

Migrating vovserver to a New Host

The failover mechanism provides the underpinnings of a convenient user CLI command that can be used to migrate vovserver to a new host:

```
ncmgr rehost -host  NEWHOST
```

The specified NEWHOST must be one of the hosts eligible for failover of vovserver.

Crash Recovery Mode

Crash Recovery Mode is activated the next time the server is restarted, if the server was not shut down cleanly. Crash Recovery Mode is part of the Failover Server capability of VOV, which is mainly used in Accelerator. This capability allows VOV to start a new server to manage the queue of a server which has crashed unexpectedly.

When you shut down an FlowTracer instance cleanly using the `ncmgr stop` command, the server will save its database to disk just before exiting. When the server is restarted, it will read the state of the trace from disk, and immediately be ready for new work.

Sometimes the vovserver will be stopped unexpectedly, such as due to a hardware problem like a machine crash or memory exhaustion. In such cases, the server will not have a chance to save the project database before terminating.

- In VOV, the main concern is usually the state of the trace, which stores the status all the jobs in your project.
- In Accelerator, there is no trace, and the important thing to preserve is the state of the queue, so jobs do not lose their position and need to be re-queued in the case of a server crash.

Journal Files

The vovserver keeps crash recovery journal files of the events that affect the state of the server. These 'CR' files are flushed whenever the trace data are saved to disk. During crash recovery, the vovserver first reads the last saved state of the trace from the disk data, then applies the events from the CR files.

Crash Recovery Restart

When the server is next restarted after such a crash, the server enters what is called Crash Recovery Mode, which usually lasts about two minutes, but may take longer if the CR files are very large. During this period:

- The server waits for vovtaskers with running jobs to reconnect.
- No jobs are dispatched.
- The server does not accept VOV or HTML TCP connections from `vovsh` or browser clients.
- At the end, the server performs a global sanity check.
- A crash_recovery_report <timestamp> logfile is written. It logs any jobs lost by the crash recovery sequence.

If the server is properly shut down, the next time it restarts, it will not enter Crash Recovery Mode, and will be immediately functional.

Note: If the sanity check command cannot connect, the server is still recovering its state from the DB and CR files. Check the size of the `vnc.swd/trace.db/CR*` files.

Example commands:

```
% vovproject enable <project-name>
% vovproject sanity
```

Advanced Information

Tasker: vtk_tasker_set_timeleft

This procedure is used in the context of time-variant tasker resources. It takes a single non-negative argument, which is interpreted as the maximum expected duration for a job to be dispatched to the tasker.

For example, if a tasker does not want to accept jobs longer than 2 minutes, it will say:

```
namespace eval VovResources {
    proc SmallJobsOnly {} {
        vtk_tasker_set_timeleft  2m
        return "@STD@"
    }
}
```

The argument of `vtk_tasker_set_timeleft` is one of:

- The string "UNLIMITED", meaning that the tasker accepts jobs of any length
- A positive integer
- A time specification
- The number 0, in which case the tasker is effectively suspended, because it does not accept any job

```
...
# -- During the day, suspend the tasker.
vtk_tasker_set_timeleft  0
...
```

Quantization of "timeleft" and "xdur"

For efficiency of the server-tasker messaging interface, all finite values passed to `vtk_tasker_set_timeleft` are silently quantized to the next largest level according to the following table:

Time Left	Quantized value
0-60s	Round up to the next multiple of 5s
60s-30m	Round up to the next multiple of 1m
30m-1h	Round up to the next multiple of 2m
1h-3h	Round up to the next multiple of 5m
3h-6h	Round up to the next multiple of 10m

Time Left	Quantized value
6h+	Round up to the next multiple of 30m

Troubleshoot Taskers

Troubleshooting TASKERS on UNIX using rsh

If you cannot connect taskers to a server, check your license file with `rlmstat`. The total number of taskers you can connect to VOV servers is limited by the license.

To start a tasker on a remote machine, `vovtaskermgr` uses `rsh` as determined by `vovrsh`. However, a problem with your account setup can prevent remote taskers from starting. The following tests check if your account is setup to run `rsh` correctly:

-
  ```
  % rsh hostname date
  ```

  This test should give you the date without asking for a password and without any other message. If a password is required, adjust the `.rhosts` file in your home directory. If any extra message is displayed, consider making the appropriate changes in your `.cshrc` file to eliminate them.

-
  ```
  % rsh hostname vovarch
  ```

  This test should give you the platform name for the remote host, nothing else. If this message does not appear, verify that your `.cshrc` file is sourcing the `.vovrc` file, even for non-interactive shells. (A common trick in `.cshrc` design is to exit early when it is discovered, by testing for existence of the variable $prompt, that the shell is non-interactive. If the exit keyword exists in your script, move the sourcing of `.vovrc` before it.)

-
  ```
  % rsh hostname
  ```

  (A login message appears at this point.)

  ```
  % vovtasker
  ```

  (The usage message from vovtasker appears.)

  If this test succeeds and the previous test does not, then your PATH variable is incorrectly set in `.login`, while it is more appropriate to set it in the `.cshrc` file. You must correct this error before you can use VOV effectively.

-
  ```
  % vovtaskermgr TEST host1
  ```

  This tests if rsh works for a list of selected hosts. A host can pass this test and still not qualify as a tasker. This can happen, for instance, if the host does not mount the file system where the project data are stored.

-
  ```
  % vovtaskermgr START -slow
  ```

This shows in detail how taskers are started. The verbose output may help you debug the problem with starting taskers.

Troubleshooting TASKERS on UNIX using ssh

Many organizations are now switching to SSH for security reasons. If you also want to use SSH instead of RSH you need to do the following:

```
% vovsshsetup
```

Then in the `taskers.tcl` file, use the option -rshcmd ssh:

```
vtk_tasker_set_default -rshcmd "ssh"
```

Now do the tests to see the rest of the setup:

```
% ssh hostname date
% ssh hostname vovarch
```

Manage Remote Taskers Without SSH/RSH Capabilities

The program `vovtsd` is a daemon written as a Tcl script that runs using the VOV `vtclsh` binary. This daemon can be used also to start various types of agents on any type of Windows or UNIX machine.

In Windows, `vovtsd` is started from the command line and then runs in a Windows `cmd` shell. Each vovserver connects to `vovtsd` via TCP/IP to start the `vovtasker` process.

vovtsd

This utility listens for requests to launch taskers for various projects, but always for the same user. The requests typically come from `vovtaskermgr`.

Usage

```

vovtsd: Usage Message

  VOVTSD: Vov Tasker Service Daemon
      This utility listens for requests to launch taskers for
      various projects, but always for the same user.
      The requests typically come from vovtaskermgr.

  USAGE:
      % vovtsd [OPTIONS]

  OPTIONS:
      -v                      -- Increase verbosity.
      -h                      -- Print this help.
      -debug                  -- Generate verbose output.
      -help                   -- This message.
      -normal                 -- Start a normal daemon (for current user)
```

```
    -expire <TIMESPEC>     -- Exit from vovtsd after specified time.
    -user  <user>          -- Specify the user that should be impersonated.
                              vovtsd computes the port number by
                              hashing the user name.
    -port  <n>             -- Specify port to listen to.
    -requireauth <n>       -- Require key-based auth from clients.
                              Client public keys need to be set in
                              VOVTSD_USERKEYS env var if enabled.
    -userkeys "<key1> <key2> ..." -- Specify a list of public keys that are
                                     allowed to authenticate the client.

EXAMPLES:
    % vovtsd -normal
    % vovtsd -port 16666
    % vovtsd -user john -port 16000
```

Start a Remote vovtasker with vovtsd

To use `vovtsd` on Windows, follow these steps:

1. Start a command shell on the Windows workstation as the user who is supposed to run the taskers. See below if this user needs to be different from the user logged in on the screen.
2. Set up the shell to use VOV with the `vovinit` command. This sets the needed environment variables, including PATH.

   ```
   c:\temp> \<install_path>\win64\bat\vovinit
   ```
3. Mount all filesystems using the appropriate drive letter; these need to agree with the values of serverdir and vovdir in the `taskerRes.tcl` file for the VOV project.
4. Start `vovtsd`, possibly using a new window. The -normal option says to use a TCP/IP port calculated from the username. You may specify the port explicitly by using the -port option.

   ```
   c:> start vovtsd -normal
   ```

Run vovtsd as a Different User

The vovtasker started by `vovtsd` will run as the user running `vovtsd`. If you need for this to be different from the user logged in at the keyboard and screen, you have several options.

On Windows 2000 and Windows XP, you can use the `runas.exe` command included with the operating system. For example, on Windows XP, logged in as 'user1', you could start a command shell using:

```
C:\temp> runas /user:domain-name\username cmd
```

Note: You may need to mount the filesystems for that user. On Windows NT, the drive letters are shared, but on later versions, each user can mount a different filesystem on a given drive letter.

vovtsd on UNIX

UNIX: Start a Remote Tasker with vovtsd

The program `vovtsd` can also be used on UNIX. In most cases, you may want to use built-in operating system commands like `rsh` or `ssh`.

You must change the `taskers.tcl` file to set the -vovtsdport variable to the port number used by your `vovtsd` daemon. For example, if `vovtsd` is running on port 16001, then your `taskers.tcl` file needs to specify the port number and a number of directories and files **as seen by the remote host**:

```
vtk_tasker_set_default -vovtsdport 16001           -vovdir     $env(VOVDIR)    -
serverdir
/home/john/vov  -logfile    /home/john/vov/$env(VOV_PROJECT_DIR).swd/logs/taskers/
@TASKERNAME@/log  -executable vovtaskerroot
```

The `vovtaskermgr` command tries to start a remote tasker first with `vovtsd` and then with either `rsh` or `ssh`. Which remote shell command is tried depends on which you have configured with the -rshcmd option of `vtk_tasker_define` in the `taskers.tcl` file.

Multi-User Altair FlowTracer

FlowTracer now supports a "multi-user" mode of operation, as a Beta feature. In the multi-user mode, many different users can share the same project, in terms of modifying a flow as well as running different compoments of the flow.

Multi-User Mode

Each job in the flow has an owner. In the traditional single user model of FlowTracer, this owner is the same as the administrator of the FlowTracer project. However, in the multi-user mode of operation, different jobs in the flow could be owned by different users. However, each user can modify or run only jobs owned by him/her. More details on how to set this up and how to run follow later in this section.

vovusergroup, UNIX Groups, and LEADER Roles

The key foundation of multiple users sharing a single FlowTracer project is the concept of VovUserGroups. All users who are intended to share a project with one another are added into the same vovusergroup. This may be accomplished by adding all members of a UNIX group to the vovusergroup using the -unix option of the `vovusergroup populate` command, or users may be added manually. However, all users must belong to the same UNIX group since the UNIX level file permissions are based off of the grown ownership of folders and files.

Once the vovusergroup has been created, it is assigned LEADER privileges in the security.tcl file.

The ACL used to allow appropriate privileges for the various multi-user operations are based upon the LEADER role.

Multi-User Mode Setup: Project Owner

All users who are expected to share the FlowTracer project must belong to a common UNIX group.

Note: The IT department should create a UNIX group, if one doesn't already exist, that includes all the users who need to work on this shared project. (For these instructions, let's use a group called "designers".)

The steps that follow should be completed by the project owner.

1. The project owner will create a new project, ensuring that he/she is using the designers UNIX group as the default group. This can be ensured by running the UNIX command `newgrp`:

```
% newgrp designers
% source <installation_directory>/<version>/<platform>/etc/vovrc.csh
% vovproject create <projectname>
% vovproject enable <projectname>
```

2. Create a vovusergroup called "designers" from the UNIX group "designers".

This will add all the users of the UNIX group "designers" as members of the vovusergroup "designers".

```
% vovusergroup populate designers -unix designers
```

3. Assign the LEADER role to the vovusergroup designers. For this, modify the `security.tcl` file to include the following line:

```
vtk_security -group designers LEADER +
```

4. Set the environment variable VOV_FT_MULTIUSER to tell `vovlsfd` that FlowTracer is running in multi-user mode.

   In this mode, `vovlsfd` will only accept and submit jobs for the user who started this instance of `vovlsfd`.
5. Restart the project so all the new settings will take effect.
6. Start `vovlsfd` to enable taskers to submit jobs to LSF.
   The project is now ready for use by other members of the designers group.

Multi-User Mode Setup: Leaders

The Leaders are the members of the designers group.

1. Change your current default UNIX group to be designers.

```
% newgrp designers
```

2. Enable the multi-user project.

> **Note:** Since you are enabling a project owned by a different user, you will need to use the -u flag on the `vovproject enable` command line to specify the user that the project is owned by.

```
% source <installation path>/common/etc/vovrc.csh
% vovproject enable -u <owner> <projectname>
% vovconsole
```

3. Set the environment variable VOV_FT_MULTIUSER to tell `vovlsfd` that FlowTracer is running in multi-user mode.
4. Start `vovlsfd` to enable taskers to submit jobs to LSF. (Note, each LEADER will need to start his/her own copy of `vovlsfd` from vovconsole.)
   Now you can view the jobs created by other users, as well as create your own jobs

Multi-User Mode Operations

To do anything with a job created by a different user of the group, you will need to take ownership of the job first.

1. To take ownership using the GUI:
   a) Double click on the job in vovconsole to open it in Node Editor.
   b) Switch to the **Execution & Impact** tab.
   c) Check the **Take Ownership** checkbox and save.
   You will now become the new owner of the job. As the owner, you may modify the job (change command, resources, etc), and run or invalidate the job.

> **Note:** you can only take ownership of jobs that are not currently running. Also, if the job has been locked by the owner (described below), you will not be able to take ownership.

2. To take ownership using the VTK API:
   a) Use the command:

```
% vtk_transition_chown_to_me <jobId>
```

The owner of a job has the ability to lock the job so no other LEADER will be able to take ownership of the job.

3. To lock ownership via the GUI:
   a) Double click on the job in vovconsole to open it in Node Editor.
   b) Switch to the **Execution & Impact** tab.
   c) Uncheck the **Allow LEADERs ownership** box and save.

4. To lock ownership using the VTK API:
   a) Use the command:

```
% vtk_acl_op <jobId> DELETE LEADER CHOWN
```

5. To lock ownership using the command line:
   a) Use the command:

```
% vovacl -delete -agent LEADER -actions CHOWN <jobId>
```

   b) To lock all the jobs in a set, use the `vovacl` command:

```
% vovacl -delete -agent LEADER -actions CHOWN <setID | setName>
```

IT Administrator Topics

SSH Setup

If you use SSH to login onto certain machines in your LAN that you want to use as taskers, you need to setup SSH so that it will not ask for your password.

You can do that using the utility `vovsshsetup`:

```
% vovsshsetup
```

Test the setup by running simple commands on the remote machine:

```
% ssh <remotemachine> date
```

Disable Host Checking for Automatic Tasker Startup

If you use SSH to start many taskers, you may find that for each new tasker you want to start you get this question:

```
The authenticity of host 'hs123 (10.11.12.123)' can't be established.
RSA key fingerprint is dd:af:39:92:c0:72:83:6b:48:25:96:4b:0e:e4:8c:76.
Are you sure you want to continue connecting (yes/no)?
```

To avoid that message and having to type 'yes' for each new host you want to connect, you can do the following:

- Edit the file `~/.ssh/config`
- Add the statement `StrictHostKeyChecking no` to the file

Network Conditioning

Each computer has its own clock. Clocks on different computers in the same network might not be synchronized. The system administrator should keep all clocks in the network synchronized.

Synchronizing Networks

The recommended method to keep the clocks synchronized is with the Network Time Protocol (NTP), available on many modern OS's, which can keep the clocks synchronized to within a few milliseconds.

Alternatively, refer to the following table to find the proper synchronization command depending on the operating system:

OS	Tool	Example
Linux	rdate	`rdate -s <host>`
Windows	net time	`net time \\<host> /set /y`

OS	Tool	Example
	Tardis™	See https://www.mingham-smith.com/tardis.htm

Whenever a vovtasker connects to the vovserver, the delta between the clocks of the two hosts is computed. Thereafter, the procedure is repeated about every 10 minutes, in order to catch clock drifts that are common in almost all computers.

File System Offsets

The file system that contains the design files may reside on a machine that is not the same as the one where the vovserver is running. The clocks in the two machines may be different, resulting in an offset. These offsets can be easily avoided. Here we show what can be done to work with an unsynchronized network.

Measuring the Offset

The complete list of known offsets is available in the Filesystems page.

To check the offset of a particular file system:

1. Login onto the machine that is running the vovserver
2. Connect to a running project
3. Change to a writable directory in the filesystem you want to test
4. Use the command `vovguru` as follows:

```
% vovguru -T .
vovguru message [11:16:49]: Filesystem /name/of/filesystem seems OK (deltaT=0)
```

The program `vovguru` writes a file in the current directory and computes *deltaT* as the difference between the file timestamp and the machine clock.

In this example, there is no offset. Otherwise, you would get a warning and *deltaT* will have a value different from 0. Acceptable offsets may be in the range of +/- 4 seconds. Larger offsets indicate a poorly managed network, and the problem should be corrected by your system administrator, rather than trying to work around it.

Dealing with the Offset

You have the following options:

- The best option is to synchronize the network.
- The second option is to allow a tolerance in the comparison of timestamps, which is accomplished by modifying the `policy.tcl` file.
- The least attractive option, if the offset is **acceptably small**, consists of adding add appropriate delays in the job firing and execution procedures. Notice that the delays affect every job, and the entire retracing system becomes slower than it could be if the clocks had been synchronized.

The file system clock is running ahead of the server

If the filesystem clock is ahead of the vovserver, as indicated by a *positive deltaT* value, sequences of jobs are likely to fail because the timestamps of intermediate input files seem more recent than the start date of the job. The failed jobs, if fired again, will typically succeed, because the files have had time to age. To avoid this kind of failures, it is sufficient to delay the firing of the jobs by setting the environment variable VOV_DELAY_FIRE. This variable, whose default value is 0, is used by vovtasker to delay job firing.

For example, if the time offset is 2 seconds, you can add the following line to the `setup.tcl` file:

```
setenv VOV_DELAY_FIRE 2
```

and then you must restart the taskers.

The file system clock is running behind the server

If the filesystem clock is running late with respect to the clock in the vovserver, as indicated by a *negative deltaT* value, it is likely that short jobs will fail because the timestamp of some outputs is younger than the job start time, giving the impression that the job has not had any effect on those outputs. The solution is to introduce a delay between the official start of the job, which is the time the job is added to the design trace, and the actual processing by the job. This is accomplished with the environment variable VOV_DELAY_BEGIN, whose default value is 0.

For example, if the time offset is 1 second, you can add the following line to the `setup.tcl` file:

```
setenv VOV_DELAY_BEGIN 1
```

Then you need to restart the taskers and source the `setup.tcl` file again in your current shell.

NFS and VOV

Before you can use NFS, be it as server or client, you must make sure your kernel has NFS support compiled in.

NFS on Linux Only

Newer kernels have a simple interface on the proc filesystem for this, the `/proc/filesystems` file, which you can display using `cat`. If `nfs` is missing from this list, you have to compile your own kernel with NFS enabled, or perhaps you will need to load the kernel module if your NFS support was compiled as a module.

NFS Mounting Options

There are a number of additional options that you can specify to mount upon mounting an NFS volume. These may be given either following the -o switch on the command line or in the options field of the `/etc/fstab` entry for the volume. In both cases, multiple options are separated by commas and must not contain any whitespace characters. Options specified on the command line always override those given in the `fstab` file. The following is a partial list of options you would probably want to use:

Option	Description
`rsize=n and wsize=n`	These specify the datagram size used by the NFS clients on read and write requests, respectively. The default depends on the version of kernel, but is normally 1,024 bytes.
`timeo=n`	This sets the time (in tenths of a second) the NFS client will wait for a request to complete. The default value is 7 (0.7 seconds). What happens after a timeout depends on whether you use the hard or soft option.
`retrans=n`	The number of minor timeouts and retransmissions that must occur before a major timeout occurs. The default is 3 timeouts. When a major timeout occurs, the file operation is either aborted or a "server not responding" message is printed on the console.
`hard`	Explicitly mark this volume as hard-mounted. This is on by default. This option causes the server to report a message to the console when a major timeout occurs and continues trying indefinitely (The client will continue to attempt the NFS file operation indefinitely if the operation fails). Hard mounts should be used when the server and the link to the server are known to be reliable. Hard mounts with the interruptible option enabled is the recommended method of mounting remote filesystems.
`soft`	Soft-mount (as opposed to hard-mount) the driver. This option causes an I/O error to be reported to the process attempting a file operation when a major timeout occurs (timeo option). Retries NFS file operation n times (retrans option) as set by the number of retries before reporting error option; returns error if no server response in n tries. Soft mounts are recommended for filesystems whose servers are considered unreliable, or if the link is slow. Unlike spongy mounts, soft mounts may time out during read and write operations.
`spongy`	Sets soft semantics for `stat`, `lookup`, `fsstat`, `readlink`, and `readdir` NFS operations and hard semantics for all other NFS operations on the filesystem. Spongy mounts are preferable to soft mounts because spongy mounts will not time out during read and write operations. They are recommended for slow, long-distance, or unreliable links, and for unreliable servers.

Option	Description
intr	Allow signals to interrupt an NFS call. Useful for aborting when the server doesn't respond.

Except for `rsize` and `wsize`, all of these options apply to the client's behavior if the server should become temporarily inaccessible. They work together in the following way: Whenever the client sends a request to the NFS server, it expects the operation to have finished after a given interval (specified in the timeout `timeo` option). If no confirmation is received within this time, a so-called minor timeout occurs, and the operation is retried with the timeout interval doubled. After reaching a maximum timeout of 60 seconds, a major timeout occurs.

By default, a major timeout causes the client to print a message to the console and start all over again, this time with an initial timeout interval twice that of the previous cascade. Potentially, this may go on forever. Volumes that stubbornly retry an operation until the server becomes available again are called hard-mounted. The opposite variety, called soft-mounted, generate an I/O error for the calling process whenever a major timeout occurs. Because of the write-behind introduced by the buffer cache, this error condition is not propagated to the process itself before it calls the write function the next time, so a program can never be sure that a write operation to a soft-mounted volume has succeeded at all.

Whether you hard- or soft-mount a volume depends partly on taste but also on the type of information you want to access from a volume. For example, if you mount your X programs by NFS, you certainly would not want your X session to go berserk just because someone brought the network to a grinding halt by firing up seven copies of Doom at the same time or by pulling the Ethernet plug for a moment. By hard-mounting the directory containing these programs, you make sure that your computer waits until it is able to re-establish contact with your NFS server. On the other hand, non-critical data such as NFS-mounted news partitions or FTP archives may also be soft-mounted, so if the remote machine is temporarily unreachable or down, it doesn't hang your session. If your network connection to the server is flaky or goes through a loaded router, you may either increase the initial timeout using the `timeo` option or hard-mount the volumes. NFS volumes are hard-mounted by default.

Hard mounts present a problem because, by default, the file operations are not interruptible. Thus, if a process attempts, for example, a write to a remote server and that server is unreachable, the user's application hangs and the user can't do anything to abort the operation. If you use the `intr` option in conjunction with a hard mount, any signals received by the process interrupt the NFS call so that users can still abort hanging file accesses and resume work (although without saving the file).

Allows the user to kill a process that is hung while waiting for a response on a hard-mounted filesystem. This option is useful if the server or the connection to the server is known to be slow or unreliable. It is recommended to always have the `intr` option on. A keyboard interrupt is configured by entering `stty intr key` where `key` is the keyboard key you wish to use to issue an interrupt.

NFS that are mounted rw should use the `hard` option (plus eventually the `intr` one).

Do not specify either `soft` and `intr` options when mounting NFS filesystems that are writable or that contains executable files.

Using PAM (Pluggable Authentication Modules) for Browser Authentication

Login using Web Browsers

For CLI and GUI commands, the user is already authenticated by the operating system login, and VOV programs are run as the logged-in user.

The `vovserver` program natively supports HTTP for VOV's browser-based user interface. When a browser connects to the vovserver, the user may wish to authenticate as one other than the user running the browser program on the remote host. Also, remote users can have complete administrative control of the host where the browser runs, and be spoofing another user's login name. Hence, VOV asks for a username and password, and uses these to authenticate the user on the host where the `vovserver` runs.

Once a user has authenticated with a given login name, their VOV privilege level with respect to VOV operations is determined by the `security.tcl` configuration file for that vovserver. Please refer to VOV Security.

Authentication Mechanisms

The `vovserver` is a PAM-aware application: its authentication may be changed without recompiling the program. On Linux and MacOS-X, the `vovserver` uses `vovpamauth` which in turn uses PAM to authenticate users.

On other platforms, `vovserver` uses the regular `crypt()` method to accept the provided password.

Authentication Control

To enable or disable PAM, you can do so with the variable *`config(enablePam)`* in the `policy.tcl` file.

```
# force PAM to be used when available
set config(enablePam) 1
```

Alternatively, you can control the use of PAM with `vovsh` and `vtk_server_config`. You must select which vovserver to act on by using the `vovproject enable` command first.

```
% vovproject enable some-ft-project
% vovsh -x 'puts [vtk_server_config enablepam 1]'
```

Note: When PAM is enabled, the `vovpamauth` program scans, at startup, the directory `/etc/pam.d`, searching in order for files named 'vovauth', 'ftauth', 'system-auth', and 'su'.

The first one found determines the PAM service name used by the `vovpamauth`.

```
# Example of a vovauth file.  Only the auth part is required.
auth       sufficient     pam_rootok.so
auth       required       pam_opendirectory.so
```

Known Issues

Problem: on Windows 2000, the vovserver Generates WSAerror 10054

This is a known problem in Windows 2000. See article 263823 in the Microsoft Knowledge Base, which recommends that you install Service Pack 2.

Health Monitoring and vovnotifyd

You can set up basic Altair Accelerator health monitoring tests with mail notification. When the Accelerator system gets into any of your defined "unhealthy" conditions, a list of configured users will receive alert email notifications.

Tests are provided that check for the following conditions:

- Long jobs that are stuck: stuck jobs do not use any CPU
- Someone has jobs waiting in queue for too long
- Any user has an unusually high ratio of failed jobs
- Any host(s) fails all jobs
- The server size (and other server related parameters) is growing
- There are too many out of queue jobs (for Allocator (also known as MultiQueue) setup only)

The checks are all procedures of which the name starts with "`doTestHealth`". These checks are defined in one or more files of the following files:

Role	Location	Notes
Global	`$VOVDIR/tcl/vtcl/vovhealthlib.tcl`	Part of distribution
Site specific	`$VOVDIR/local/vovhealthlib.tcl`	Optional
Project specific	`PROJECT.swd/vovnotifyd/vovhealthlib.tcl`	Optional

Configuring Health Monitoring

By default, all checks defined in the `vovhealthlib.tcl` files are enabled.

If you want to change parameters in the health checks, you need to change the `config.tcl` file in the `vovnotifyd` directory.

The following table contains an example of `vovnotifyd/config.tcl`:

```
set NOTIFYD(server)          "tiger"
set NOTIFYD(port)            25
set NOTIFYD(sourcedomain)  "mycompany.com"

set admin  "dexin"
set cadmgr "john"

#
# Check if we have long stuck jobs. Check every 1 minute. If we have such
# jobs, send alert emails to the owner of the job (@USER@) and
# admin (here "dexin").
#
# Definition of "long stuck job": has been running at lease 10 hours and but has used
 no
# more than 20 seconds CPU time in total.
#
registerHealthCheck "doTestHealthLongJobsNoCpu -longJobDur 10h -minCpu 20"  -
checkFreq 1m  -mailFreq  1d  -recipients "@USER@ $admin"

#
# Check if we have jobs stuck, i.e., jobs that are not burning any CPU at all. And
 this situation
# has been persistent for at least 10 minutes.
#
# Check every 1 minute. If we have such jobs, send alert emails to the owner of the
# job (@USER@) and admin (here "dexin").
#
# This check is similar to doTestHealthLongJobsNoCpu but will be quicker to detect
 stuck jobs.
#
registerHealthCheck "doTestHealthJobStuck -maxNoCpuTime 10m"  -checkFreq 1m  -
mailFreq  1d  -recipients "@USER@ $admin"

#
# Check if any user has too many failed jobs. Check every 30 minutes. If we
# have such users, send alert emails to the owner of the job (@USER@),
# admin (here "dexin") and cadmgr ( here "john" ).
#
# Definition of "too many failures": a user has at least 1000 jobs in NetworkComputer
# with at least 90% failures
#
registerHealthCheck "doTestHealthTooManyFailures -minJobs 1000 -failRatio 0.9"  -
checkFreq 30m  -mailFreq  1d  -recipients "@USER@ $admin $cadmgr"


#
# Check if the server size is growing. Check other server related parameters as
# well, including number of jobs, number of queued jobs, etc.
#
# For everything that is checked, if the number grows over 60.0% compared to last
 time
# it is checked, send alert emails to the admin (here "dexin")
# and cadmgr ( here "john" ).
#
# Also send alert emails if the number of files is 5.0 times or more than the number
# of jobs.
#
# Check every 2 hours and send such alert emails once a day (1d).
#
registerHealthCheck "doTestHealthServerSize -filejobsRatio 5.0 -warnPercent 60.0"  -
checkFreq 2h  -mailFreq  1d  -recipients "$admin $cadmgr"

#
# Check if some user has jobs sitting in the queue for too long.
# Check every 2 hours.
```

```
#
# If we find such users, send alert emails to the user
# and cadmgr ( here "john" ). Send such alert emails once a day (1d).
#
# Definition of "waiting for too long": none of jobs in one category(bucket)
# get dispatched in the last 4 hours.
#
#
registerHealthCheck "doTestHealthJobsWaitingForTooLong -maxQueueTime 4h"  -checkFreq
 2h  -mailFreq  1d  -recipients "@USER@ $cadmgr"
```

The `config.tcl` file is checked for updates at regular intervals controlled by the variable *`NOTIFYD(timeout)`*.

vovnotifyd

The `vovnotifyd` daemon performs two functions:

- Sends mail notification based on job events, according to the MAILTO property of the jobs; (Altair Accelerator only)
- It performs periodic system health checks, and sends email when it discovers issues. (Monitor and Accelerator)

There are a number of predefined system health check procedures included with the Altair Accelerator. You can also write health check procedures to monitor specialized conditions at your site. See below.

Configuring vovnotifyd

Table 2: Summary information for vovnotifyd

Config files	`vnc.swd/vovnotifyd/config.tcl`
	`vnc.swd/vovnotifyd/config_smtp.tcl`
	`vnc.swd/vovnotifyd/config_export.tcl`

Info file	`vnc.swd/vovnotifyd/info.tcl`

Auxiliary files	`$VOVDIR/tcl/vtcl/vovhealthlib.tcl`
	`$VOVDIR/local/vovhealthlib.tcl`
	`vnc.swd/vovnotifyd/vovhealthlib.tcl`

The easiest way to configure notification is from the Admin page of the browser interface. Click on the **Daemons** item on the left-hand menu, then click **config** in the row for **vovnotifyd**.

To manually configure `vovnotifyd`, you need to create the directory `vovnotifyd` inside of the server working directory (`.swd`) and copy the configuration file template, `$VOVDIR/etc/config/vovnotifyd/`

`config.tcl` into the newly-created `vovnotifyd` directory. The configuration template will need to be modified to match the settings of your mail server environment.

```
% cd `vovserverdir -p .`
% mkdir vovnotifyd
```

File: `$VOVDIR/etc/config/vovnotifyd/config.tcl`

```
# Notification configuration file.
# Should be placed in the vovnotifyd directory of the .swd.
# All settings are required unless specified otherwise.
# Unused optional settings should be commented out.

# Create an e-mail address map, stackable, optional
addUserToEmailAddressMap  rtdamgr john@mydomain.com

### Altair Monitor-specific settings
# See notification configuration documentation in Altair Monitor Admin Guide
# ConfigureTag     TAG OPTION VALUE
# ConfigureFeature FEATURE OPTION VALUE

### Examples:
# ConfigureTag MGC -poc { john mary }
# ConfigureFeature EDA/MATLAB -longcheckout 2d -userlongcheckout john 1w -mincap 5 -
triggerperc 90
# ConfigureFeature SIMULINK -poc bob -mincap 10 -triggeruse 12
```

To start the daemon, either use `nc cmd vovdaemonmgr start vovnotifyd` or start it manually from within the `vovnotifyd` directory:

```
% vovproject enable vnc
% cd `vovserverdir -p vovnotifyd`
% vovnotifyd
```

Autostart vovnotifyd

In the directory `vnc.swd/autostart` create a script called `start_vovnotifyd.tcl` with the following content:

```
% cd `vovserverdir -p .`
% mkdir autostart
% cp $VOVDIR/etc/autostart/start_vovnotifyd.tcl  autostart/start_vovnotifyd.tcl
```

Configure Email Addresses

You can use the `config.tcl` file to set the email addresses to be used for each user. You can choose one of the following methods:

- Call `addUserToEmailAddressMap USERNAME EMAIL`
- Override the entire procedure `getEmailAddress`

```
# Fragment of vovnotifyd/config.tcl file.

# Method 1.
addUserToEmailAddressMap john  John.Smith@my.company.com

# Method 2. Assume we can get an address from LDAP
#            The LDAP subsystem needs to be configured.
proc getEmailAddress { user } {
    set email [VovLDAP::getEmail $user]
    if { $email != "" } {
       return $email
    } else {
       return $user
    }
}
```

Write Localized Health Checks

The `vovnotifyd` daemons runs in the `vovsh` binary, so all the VTK API procedures are available to you.

The standard checks procedures are defined in the file `$VOVDIR/tcl/vtcl/vovhealthlib.tcl`.

The health check procedures are loaded using a search path. First, they are loaded from the file given above, then from `$VOVDIR/local/vovhealthlib.tcl`, and then from `vovhealthlib.tcl` in the `vovnotifyd` working directory. This permits you to redefine health check procedures on a site-wide or project-specific basis.

Note: Any local procedure names should begin with 'doTestHealth' like the system ones; some things depend on this convention.

For local procedures to be shown by the browser UI, you need to add a line into your `vovhealthlib.tcl` file like:

```
set HEALTHLIB_PRODUCT_MAP(doTestHealthYourProcedure)        "nc"
```

The product names are those returned by the procedure `vtk_product_get_info -name`: nc, lm, lam, ft, wa

Be sure your procedures are robust and handle error conditions well, that is, by catching all `exec{}`, `open{}` and other procedures that can sometimes fail.

Note: Changes made to any of the `vovhealthlib.tcl` files will not take effect until the `vovnotifyd` daemon is restarted.

Alternate Methods of Sending Email

When direct SMTP does not work, there are other methods to send email that may.

Using a Mailer Program

The standard `sendMail` procedure checks the config file variable *`NOTIFYD(mailprog)`*, which has a default of *`SMTP`*. This may also be set to the name of a mail program that can accept a message body on its 'stdin' stream, like `/bin/mail` on UNIX. This program is also expected to have an -s option to specify the subject.

Overriding the sendMail Procedure

The procedure `sendMail` defined in the file `vovnotifydlib.tcl` uses SMTP (Simple Mail Transfer Protocol) to send email notification, by connecting directly to a mail transport agent (MTA), such as sendmail, postfix, or Microsoft Exchange. The `vovnotifyd` is only capable of simple unauthenticated, unencrypted SMTP, and may not work if the MTA is configured to require authentication or transport-level security.

If your MTA is so configured, you may need to use an alternate method to send mail. One method is to redefine the procedure daemon's `sendMail` procedure. There is also another way, see below.

```
# Overriding sendMail in the vovnotifyd/config.tcl file.
proc sendMail { recList subj msg } {
    catch {exec yourMailScript.csh $recList $subj $msg }
}
```

The script that you write can perhaps be a simple wrapper around `/bin/mail` on UNIX, or you can even exec such a mailer program directly in the `sendMail` procedure.

If the host on which `vovnotifyd` runs has a local MTA which relays to the domain's MTA which requires authentication or TLS, the local MTA may accept unauthenticated cleartext SMTP. If this is the case, you may be able direct SMPT with 'localhost' as the mailer host.

Health Checks

This is a list and short description of all built in health checks monitored by `notifyd`.

By default, all checks occur every 10 minutes.

`AllJobsFailedOnHost`	This check will check for hosts on which all jobs have failed during a given time window (default 3600)
`CheckAlerts`	This check will send a summary of all the current alerts via notification email
`CheckDownTaskers`	This check will check for taskers enabled in `taskers.tcl` but not running
`CheckJobsReqstRam`	This will alert you for jobs that have used more RAM than requested.
`CheckTaskerset`	This will check the status of running taskers.
`CheckVendorLicenseExpiration`	This will alert you when a vendor license is going to expire in a few days.
`Daemons`	This will check the daemons.
`FalseLicenceUsage`	This will alert you when jobs use licenses that have not been declared

`JobStuck`	This will find stuck jobs, which are running but burning no CPU. The default check frequency on this is 2 hours.
`JobsWaitingForTooLong`	Just as the name suggests, this reports on jobs that have been waiting for too long.
`LongJobs`	This will check for jobs that run too long.
`LongJobsNoCpu`	Like the previous check, but will check for jobs that are not using any CPU.
`RamSentry`	This will check jobs under Ram Sentry protection.
`ServerSize`	This will check the growth of vovserver.
`TooManyFailures`	As the name suggests, this will check for a user submitting too many failed jobs.
`TooManyOutOfQueueJobs`	This will check for too many jobs being run outside of Altair Accelerator.

Check All Files in a Project

When you request a run, the FlowTracer server checks the timestamp of all primary inputs in the upcone of the target.

To check all files in the up-cone, instead of only the primary inputs, we recommend the VOV_VW_CHECK_UPCONE variable, which tells the wrapper to check all files in the up-cone at runtime.

To check the timestamps of all the files in a project, including therefore also all the intermediate files, use `vovcheckfiles`.

```
vovcheckfiles: Usage Message

  DESCRIPTION:
     Check the timestamp of all files in a project.

     This is an expensive operation, so this script
     implements a semaphore to avoid concurrent checks.

  USAGE:
      % vovcheckfiles  [OPTIONS]

  OPTIONS:
      -v     -- Verbose
      -h     -- This help

  EXAMPLES:
      % vovcheckfiles -h
      % vovcheckfiles
      % vovcheckfiles -v
```

If the timestamp of any file in the project has changed, for example by being modified by an editor or other program run outside of the flow, the vovserver is notified of the change. This may cause the down-cone of the modified file(s) to be marked INVALID.

Troubleshoot vovcheckfiles

Since different hosts can see different file attributes (and even existence) due to NFS caching, if multiple instances of the command run concurrently, the vovserver could receive conflicting information. Also, checking all the files in a large project is an expensive operation. Hence, the command tries to have only one instance running.

The command uses a property VOVCHECKFILES_SEM attached to the flow (i.e. the object with VovId 1) as a semaphore, and other properties to record the time it started, and the user@host:process-id. Except for the timestamp, the properties are deleted at the end of a normal run. If `vovcheckfiles` is interrupted, the semaphore property will be left on the flow. If the semaphore count is > 1, but the timestamp is old, `vovcheckfiles` will give a message stating this and start the check, instead of exiting.

You can see the state of the properties by using:

```
% vovprop get -default "" 1 VOVCHECKFILES_SEM
```

vovcleanup

The utility `vovcleanup` is used to clean old log files from `/tmp` and from the server working directory.

```

vovcleanup: Usage Message

DESCRIPTION:
    vovcleanup  -- Cleanup log files in host and project directory.
                   Rules can be customized in server working
                   directory(swd)/cleanup.config.tcl

                   An example of cleanup.config.tcl can be found here:
                   $VOVDIR/etc/ProjectTypes/generic/cleanup.config.tcl

OPTIONS:
    -containercache     -- Cleanup container cache files.
    -containercachepath -- Path to the container cache directory.
                           Default is SWD/containers.
    -h                  -- Help usage message.
    -host               -- Cleanup log files in /tmp,/usr/tmp,/var/tmp.
    -n                  -- Show files to delete, but do not delete them.
    -proj               -- Cleanup project log and temporary files.
                           Empty profile and wave subdirectories under
                           SWD/data are also removed.
    -v                  -- Increase verbosity.

EXAMPLES:
    % vovcleanup -host
    % vovcleanup -n -proj
    % vovcleanup -proj

SEE ALSO:
    vovcleandir       -- to cleanup old log files in a directory.
    nc clean          -- to cleanup old files generated by NetworkComputer.
```

To cleanup old log files in the `/tmp` and `/usr/tmp` directory, use:

```
% vovcleanup -host
```

To cleanup log files and cache files in the project configuration directory, use:

```
% vovcleanup -proj
```

If you want to first see which files are going to be cleaned up, use the `-n` option as in:

```
% vovcleanup -proj -n
```

Cleaning rules can be modified in `cleanup.config.tcl` located in the Server Working Directory.

GUI Customization

Both the vovconsole's main GUI and its Node Editor GUI can be customized with the `gui.tcl` file.

The `gui.tcl` file is found in one, or both, of the following directories:

- `$VOVDIR/local`
- The server configuration directory (`.swd`)

`$VOVDIR/local/gui.tcl` is installation-based customizations, while `$SWD/gui.tcl` project-based customizations. `$VOVDIR/local/gui.tcl` will always be sourced, followed by `$SWD/gui.tcl` (if it exists). Knowledge of Tcl/Tk is required in order to customize the vovconsole GUI.

By running `vovconsole` with the option -showcustomize, you can see the regions of the GUI which are customizable. The following image shows the customizable regions of the main GUI.

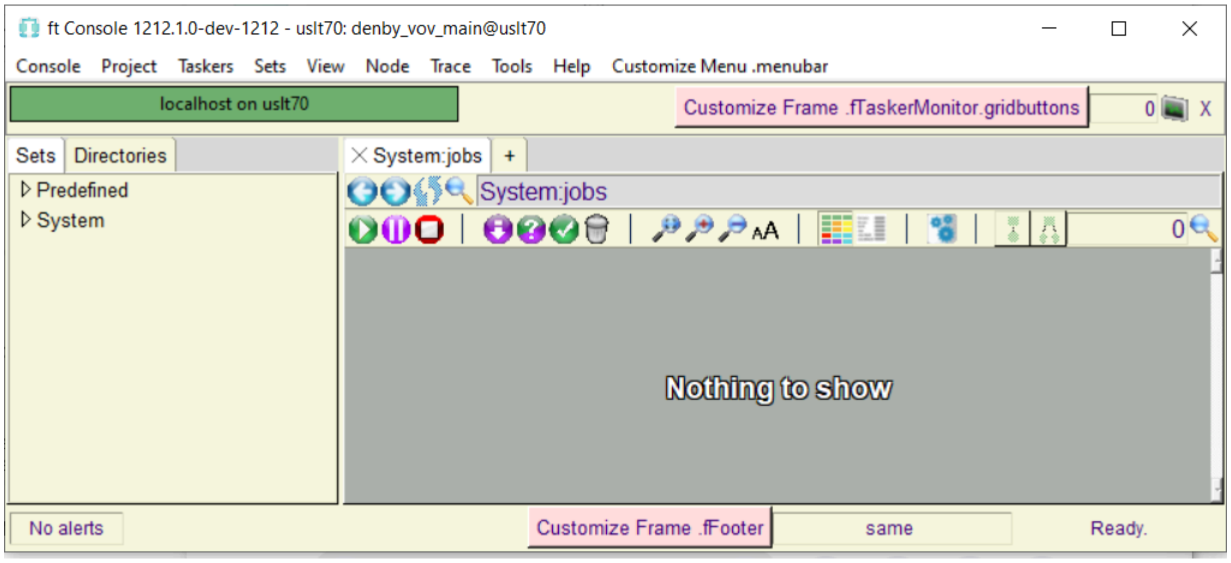

ft Console 1212.1.0-dev-1212 - uslt70: denby_vov_main@uslt70
Console Project Taskers Sets View Node Trace Tools Help Customize Menu .menubar
localhost on uslt70
Customize Frame .fTaskerMonitor.gridbuttons
Sets Directories
Predefined
System
System:jobs
Nothing to show
No alerts
Customize Frame .fFooter
same
Ready.

Figure 7:

You can see the pinkish labels containing "Customize <`TkWidgetType`> <`TkWidgetPath`>" information messages. These labels point out where customization may occur, and what Tk widget path to use. On the main GUI, both the menu (type=`menu` path=`.menubar`) and tasker monitor frame (type=`frame` path=`.fTaskerMonitor.gridbuttons`) are customizable.

Additionally, new tabs can be added to the Node Editor.

Set a Custom Title

To set a custom title, call the procedure `::VovGUI::setConsoleTitle` in the `gui.tcl` file. This procedure takes a single argument: the custom string that will be displayed at the beginning of the vovconsole title.

Implementing ::VovGUI::customize

The procedure `::VovGUI::customize` is called after many of the elements of the vovconsole have been created. You can create your version of `::VovGUI::customize` in your `gui.tcl` file and operate on the elements. For example you can add a new command in the pop-up menu, a new button along the bottom of the Node Editor, or some artwork in the menu-bar at the top. A very good knowledge of Tcl/Tk is required.

```
#
# This is a fragment of gui.tcl
#
proc ::VovGUI::customize { widget } {
    set mycompany "mycompany"
    VovMessage "Opportunity to customize window $widget" 3
    switch -glob $widget {
        ".menubar" {
            menu $widget.$mycompany -tearoff false
            $widget add cascade -label {MyCompany} -menu $widget.$mycompany -
underline 2
            $widget.$mycompany add command -label "MySettings"  -underline 0  -
accelerator 'S' -command {exec /usr/X11/bin/xterm}
 }
```

```
        ".popupNode" {
            $widget add command -label "DoSomething" -command
 "::VovGUI::MySpecialCallBack"
        }
        ".vgui-nodeviewer.fTop.toolbar" {
            # Add a button along the Node Editor's bottom toolbar
            button $widget.newButton -bd 1 -text "Report Issue" -command
 "::VovGUI::ReportIssue"
            pack $widget.newButton -side left -padx 10 -pady 2 -fill y -expand 0
        }
        "*.xleFormat" {
            ### Customize the list of formats in "list mode".
            $w insert end "@ID@ @JOBNAME@"
        }
        ".nodemenu" {
            # Customize the right click popup menu in the set browser
            $w add command -label "Do something" -command MyCallback
        }
    }
}

# Example of call back
proc ::VovGUI::MySpecialCallBack {} {
    set nodeId    $::VovGUI::currentNodeID
    set selection [::VovGUI::GetSelection $::VovGUI::currentWidget sel 0]

    puts "DO SOMETHING WITH $nodeId"
    puts "Selection = $sel"
    puts "Widget = $::VovGUI::currentWidget"
}

proc ::VovGUI::ReportIssue { } {
    # This procedure will determine which Node Editor tab is active, and will produce
    # a message box popup with the contents of the tab.  This only applies to
    # the 'Why' and later tabs.

    # Get notebook widget handle
    set nb ".vgui-nodeviewer.fTop.vwxNodeEditor.xnbMain"
    # Figure out 'active' tab
    set activeTab [$nb raised]

    # Get text contents of active tab
    switch $activeTab {
        #"node"         { set txt "" }
        #"annotations" { set txt "" }
        "why"         { set txt ["$nb.nbframe.why.xstWhy.text"       get 1.0 end] }
        "out-1"       { set txt ["$nb.nbframe.out-1.xstContent.text" get 1.0 end] }
        "out-2"       { set txt ["$nb.nbframe.out-2.xstContent.text" get 1.0 end] }
        "out-3"       { set txt ["$nb.nbframe.out-3.xstContent.text" get 1.0 end] }
        "out-4"       { set txt ["$nb.nbframe.out-4.xstContent.text" get 1.0 end] }
        "out-5"       { set txt ["$nb.nbframe.out-5.xstContent.text" get 1.0 end] }
        default       { set txt "Tab '$activeTab' functionality not supported" }
    }

    # Create message box popup with the text contents
    set answer [tk_messageBox  -icon question  -type yesno  -title "Do you want to
 submit the issue?"  -parent .vgui-nodeviewer.fTop  -message "$txt"]

    if {$answer == "yes"} {
        # Add code here to email the text to your issue tracking system
    }
}
```

Add a Logo to the Console

You can add elements to the console if you know the names of the Tk widgets that compose it. In the following example, we add a graphic element to the menu bar, which happens to be the `.menubar` widget.

```
#
# This is a fragment of gui.tcl
# EXAMPLE: Add a logo to the Menu Bar.
#
proc addMyLogo {} {
    if [winfo exists .menubar] {
        set ppm /path/to/mylogo.ppm
        set img [image create photo -file $ppm]
        .menubar add command -image $img
    }
}

# This is a bit tricky.
after 3000 { ::VovGUI::addMyLogo }
```

Change Colors of Jobs Based on Exit Status

You can use `vtk_transition_color_control` to choose the color used to represent VALID and FAILED jobs in the console, depending on the exit status. For example, to display VALID jobs that exited with status 10 using the color "pink", write:

```
# Fragment of gui.tcl
vtk_transition_color_control SET_VALID 10 pink
```

To display FAILED jobs that exited with status 255 using the color "brown", do the following:

```
# Fragment of gui.tcl
vtk_transition_color_control SET_FAILED 255 brown
```

Reference documentation for `vtk_transition_color_control` can be found in the *VOV Reference Guide*.

Choose the Editors Depending on File Types

Use the procedure `::VovGUI::addFileEditor` to describe the editor to be used for files with a given suffix. Example:

```
# Fragment of gui.tcl
::VovGUI::addFileEditor ".lef" "xterm -e vi"
```

Change Default Priorities and Retrace Flags

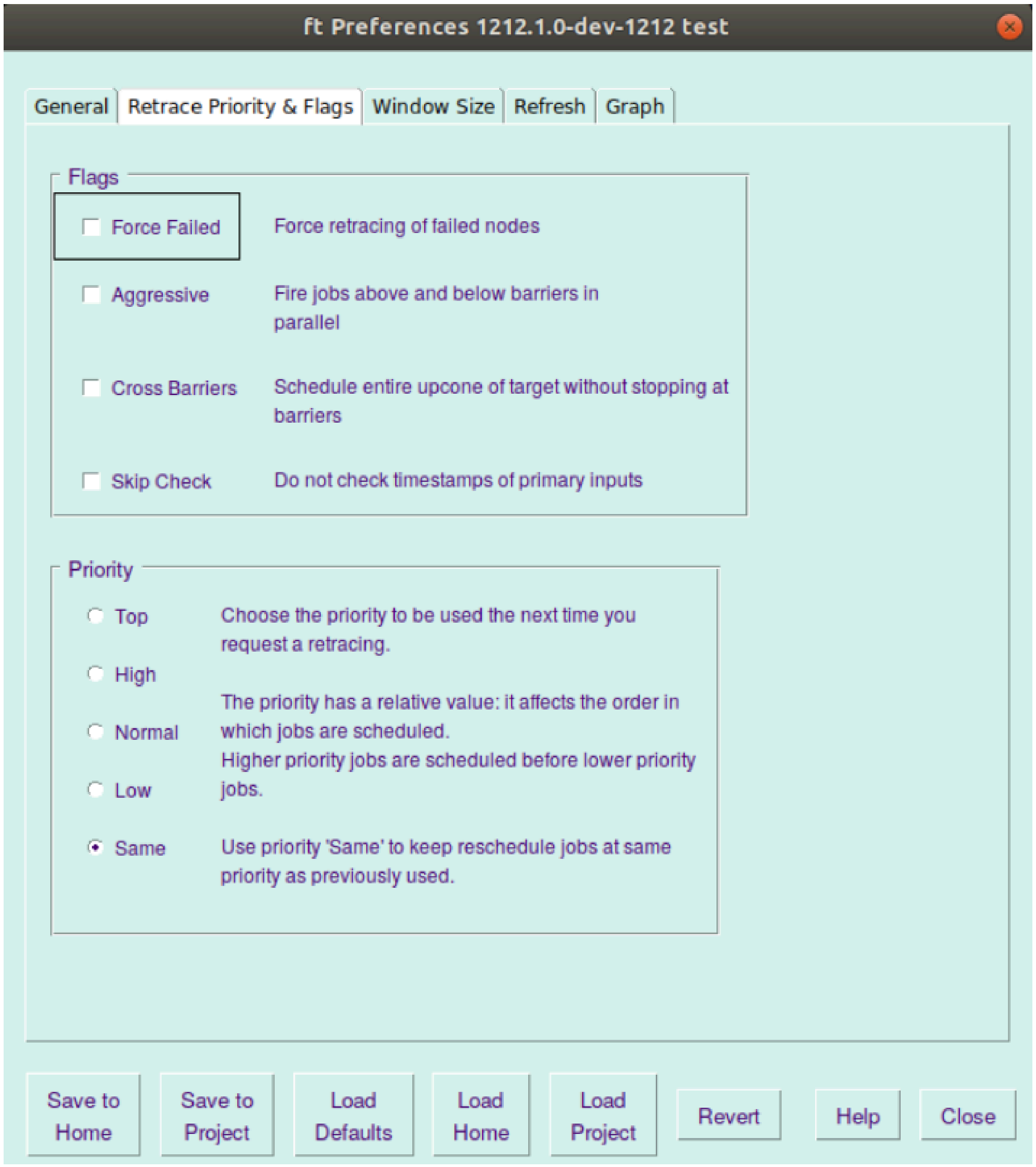
Examples for directly access the controlling variables (one for the control, one for the GUI):

```
# Controls on priority (case-sensitive)
set ::VovGUI::retrace(priority) Low
set ::VovGUI::preference(retrace.priority) LOW

# Controls on mode
# Values for retrace(mode):  force crossbarriers skipcheck aggressive or SAFE
set ::VovGUI::retrace(mode)  "crossbarriers force skipcheck"
set ::VovGUI::preference(retrace.force.failed)   1
set ::VovGUI::preference(retrace.crossbarriers)  1
set ::VovGUI::preference(retrace.aggressive)     0
set ::VovGUI::preference(retrace.skipcheck)      1

# Finish by refreshing the console
::VovGUI::UpdatePriorityFooter
```

When changing the `VovGUI::preference nn54;lr(retrace.XX)` options, you will get two Save options: one option will save the preferences to your home directory, and the other with save them to your project directory (SWD).



Flags

☐ Force Failed	Force retracing of failed nodes
☐ Aggressive	Fire jobs above and below barriers in parallel
☐ Cross Barriers	Schedule entire upcone of target without stopping at barriers
☐ Skip Check	Do not check timestamps of primary inputs

Priority

○ Top
○ High
○ Normal
○ Low
● Same

Choose the priority to be used the next time you request a retracing.

The priority has a relative value: it affects the order in which jobs are scheduled.
Higher priority jobs are scheduled before lower priority jobs.

Use priority 'Same' to keep reschedule jobs at same priority as previously used.

Figure 8:

There are two options for changing the default priorities and retrace flags behavior.

- GUI checkboxes will **not** reflect the default settings: Use `set ::VovGUI::retrace(X)`.
- GUI checkboxes **will** reflect what you have set the default to be and use that default, use `set VovGUI::preference(retrace.X)`

Either choice will result in a `vovconsole.preferences.tcl` file being created (in either home or SWD) and it is THAT file that normally would contain these preference settings.

CAUTION: If you choose to put these changes in `SWD/gui.tcl` file, the changes will override what is in the `vovconsole.preferences.tcl` file.

Node Editor

By default, the Node Editor has six tabs :

- Job
- Annotations & Properties
- Why?
- O:Stdout
- O:Stderr
- O:Log

Add Tabs in the Node Editor

To add new tabs in the Node Editor, use the procedures `::VovGUI::addOutputTab` and `::VovGUI::addInputTab` . These procedures require 3 arguments:

1. The label to be given to the tab.
2. The regular expression to match the file to be displayed.
3. A special expression to determine if the tab is to be displayed. This expression can take one of the following forms:

Argument	Effect
+	Show tab always
+toolname	Show tab only for jobs that use tool 'toolname'.
-string	Show tab only for jobs with a command line that contains the string 'string'.
TOOL~value	Show tab for jobs whose tool matches 'value'. Example `TOOL~spice`
COMMAND~value	Show tab for jobs whose command line matches 'value'.
ENV~value	Show tab for jobs whose environment matches 'value'.

Argument	Effect
STATUS~value	Show tab for jobs whose status (in string format) matches 'value'.

Here is an example of a gui.tcl file.

```
# Fragment of gui.tcl
# Show the VNC log file for all jobs that have tool equal to "vwrap"
::VovGUI::addOutputTab "vnclog"  {/vnc_logs/[0-9]+/[0-9]+\.[0-9]+$}  "-vwrap"
::VovGUI::addOutputTab "Err"     {/.*\.err$}                          "+"
::VovGUI::addOutputTab "Log"     {/.*\.log$}                          "+"
::VovGUI::addInputTab  "C-src"   {\.c$}                             "-gcc"
```

Annotations

Annotations are text strings which may be attached to VOV objects. These are typically used to provide additional information about objects in the project's graph. The text must be less than 32k characters.

You may attach annotations to any VOV element in the dependency graph. Use annotations to explain either why you did something or the meaning of a file.

Add Annotations

You may add annotations to VOV objects from the VOV console GUI, or programmatically when the flow is built. At this point, there is no support to add annotations using the browser interface.

VOV Console

Use the **Property** dialog to add annotations. Double-click on a node in the grid view, then click on the **Annotations & Properties** tab of the Node Editor after it is displayed.
An annotation is itself a VOV object with its own VovId.

Annotations in the Flow Definition File

You may attach annotations to objects in your flow at the time it is built by using the **A** statement in the Flow Definition File. Refer to FDL Procedures Reference for additional information.
Example of attaching an annotation to a job:

```
J vw cp aa bb
A "this is a simple job"
```

Annotation Support in the Tcl API

You may programmatically attach, modify, and delete annotations at any time using procedures in the `vtk_` application programming interface.

Annotation procedures	Description
vtk_annotation_count	Gets an integer count of the number of annotations on a VOV object

Annotation procedures	Description
vtk_annotation_get vovid index	Gets data of an annotation in list form
vtk_annotation_del	Deletes an annotation
vtk_annotation_modify	Modifies data of an annotation
vtk_annotation_add vovid string	Adds 'string' as an annotation to 'vovid'

Ignore Links into the DesignSync Cache

If you use DesignSync, you want to ignore links to the DesignSync cache.

This can be done by setting the variable VOV_SYNC_CACHE_DIR to the full path to the DesignSync cache. The symlinks will not be resolved to the final name, but will keep the original symlink name. The nodes are still present in the graph (they are not excluded, as with `exclude.tcl`).

For example, suppose that you have the cache directory in `/opt/local/dssc/sync_cache`, you can avoid showing the cache by setting the following environment variable before running the tools:

```
% setenv VOV_SYNC_CACHE_DIR `vovequiv -p /opt/local/dssc/sync_cache`
```

We use `vovequiv -p ...` to eliminate any symbolic links in the path of the DesignSync cache itself.

The variable VOV_SYNC_CACHE_DIR is read by the wrappers (vw, vov, ...) at runtime.

If you use multiple caches, use a colon-separated list of paths as the value of the VOV_SYNC_CACHE_DIR.

```
% setenv VOV_SYNC_CACHE_DIR /opt/local/dir1/sync_cache:/remote/.sync/cache_dir
```

Let's assume you have the following file on disk:

```
%  ls -lrt top.v
lrwxr-xr-x  1 ged  staff  126 Aug  9 14:37  top.v ->
      /opt/local/dssc/sync_cache/sfb/sfbf04ef93dg158ed-1.2-1310712769-0
```

FlowTracer will display the file as `top.v` when the VOV_SYNC_CACHE_DIR environment variable is correctly set.

The real file name `sfbf04ef93dg158ed-1.2-1310712769-0` will not be displayed.

To further debug this mechanism, you can use the `-d` option in `vovequiv` to see how symbolic links are mapped:

```
% vovequiv -d -p aaa
vovsh Aug 29 09:49:39 SYMLINK: SYNC_CACHE_DIR is '/remote/projects/sync_cache'
vovsh Aug 29 09:49:39 SYMLINKS: Replaced '/home/casotto/projects/ac8head/testing/
main/dssc_test_run/abc.txt' with '/home/casotto/projects/ac8head/testing/main/
dssc_test_run/tmp_sync_cache/1234567890-1.2-123456789CACHE_FILE1234'
```

```
vovsh Aug 29 09:49:39 SYMLINK: path '/remote/projects/
sync_cache/1234567890-1.2-123456789CACHE_FILE1234' is in CACHE_DIR
```

Special Issues

Browser Security

The browser interface requires the user to authenticate using matching login and password. The authentication remains valid until you either terminate the browser or logout.

To disable authentication, set the variable *`config(httpSecure)`* to 0 in the `policy.tcl` file.

To completely disable the HTTP interface, set the variable *`config(httpSecure)`* to "disable" or any integer other than 0 or 1 (use 2, typically). To enable the HTTP interface, use the value "enable" or 1.

As of 2016.09, web server security has been enhanced to prevent local file inclusion via a path relative to an open URL, such as /gif.

Add Custom Navigation Menu to Browser View

You can change the FlowTracer web pages viewed through the browser so there is a customized navigation menu along the top. This allows you to provide explicit links so users can access other useful web pages without needing to create bookmarks to get there.

The added navigation menu is displayed just above the bread-crumb path in the top left of the page. The navigation menu can have one or many navigation choices.

The navigation menu choices are placed on a web page as anchor tags, which are standard HTML elements to define hyperlinks. An anchor tag is composed of an anchor text and a URL. The anchor text is what is displayed on the web page as the navigation choice. The URL is the hyperlink that is followed when a user clicks on the anchor text.

Example HTML source generated for custom navigation menu:

```
[<a href="http://license.myhost.com:1721">licmon</a>]
[<a href="cgi/cgi-metrics.cgi">metrics</a>]
[<a href="URL">anchor text</a>]

Which would display as a custom navigation menu looking like this:
[<a href="">licmon<a>] [<a href="">metrics<a>] [<a href="">anchor text<a>]
```

Note: The links in this and other examples on this page are not live links.

The URL can be a path to a location on the same domain, or can hold a fully qualified path to a resource at any domain.

Define Custom Navigation Menu Items

A custom navigation menu is registered by setting the EXTLINKS property of the unique "***graph object***" node in the flow. This graph object node is the one with an id value of 1. You set the EXTLINKS property of this particular node to define the custom navigation menu. Setting the EXTLINKS property on any other node will not do anything.

The added navigation menu is defined by placing pairs of anchor texts and URLs into the EXTLINKS property. Each pair of values defines one navigation menu entry. The anchor text and URL values are separated by one or more blank spaces. The pairs of values are separated by one or more blank spaces.

```
EXTLINKS Property Value:
anchor_text URL   anchor_text URL   anchor_text URL   ...
```

where `anchor_text` is the text displayed for the menu choice and `URL` is the hyperlink to follow when the navigation menu choice is clicked.

When there is an odd number of values in the EXTLINKS property, this means that one entry is not a pair. This will lead to the situation where the last value in the list will be interpreted as anchor text for a navigation menu choice but the associated hyperlink URL will be undefined.

Use Multiple Words in a Custom Navigation Menu

If you want to use multiple words in the anchor text that is displayed as the menu choice, you must set the space between words to something that is not a blank space, because a blank space is considered a separator between the anchor text and the URL, and between the pairs.

You can use an underscore or dash between words instead of a blank space. This gives a menu choice that is readable. However, the menu choice will contain the underscore or dash.

```
EXTLINKS Property Value:
Altair_Monitor  http://mydomain.com:3866   View-Metrics  /cgi/metics.cgi

Which would display as a custom navigation menu looking like this:
    [Altair_Monitor] [View-Metrics]
```

You can use the HTML entity code for a blank character (`&nbsp;`) between words instead of a blank character. This will cause the menu choice to show a blank character between words.

```
EXTLINKS Property Value:
Altair&nbsp;Monitor  http://mydomain.com:3866   View&nbsp;Metrics  /cgi/metics.cgi

Which would display as a custom navigation menu looking like this:
    [Altair_Monitor] [View-Metrics]
```

Set the EXTLINKS Property

You can set the EXTLINKS property from the command line, or from the browser interface.

This example shows setting the property value from the command line using the `vovprop` command to define a navigation menu. The value follows the above rules for defining two navigation choices.

```
# The property is type text, is being set on node 1,
# and defines two custom navigation menu choices.
# The action "SET" can be lower case. The property name "EXTLINKS" must be upper
 case.
```

```
% vovprop SET -text 1 EXTLINKS 'License&nbsp;Monitor  http://myhost:5555   Metrics  /
cgi/metrics.cgi'
```

Set the EXTLINKS property to an empty string to remove the custom navigation menu.

```
# The property is type text, is being set on node 1, and has an empty value,
# as the technique to remove the custom navigation menu.
# The action "set" can be upper case. The property name "EXTLINKS" must be upper
 case.
% vovprop set -text 1 EXTLINKS ''
```

If you have Admin Security privilege, you can change the EXTLINKS property using a web based dialog in the browser. You can access this web based dialog either by going to the Messages page, or by clicking on the **"[Edit Message]"** navigation control that is visible to Admin Users in the bottom left of the page.

Both paths lead to a web page dialog where you can edit the EXTLINKS property. You are presented with a table for doing data entry, where the anchor text can be entered in the column called **Label** and the URL can be entered in the column called **URL**. Clicking **Update EXTLINKS Property** button submits the changes to the system.

Miscellaneous

NFS Cache

NFS caches some information about files and directories and this may cause corrupt data in some flows where multiple machines are used, because one of the machines may happen to use cached data that is not up to date, or may fail to see a file that has been just created on some other machine.

To minimize the failures caused by the NFS caches, the VOV wrapper have the ability to check that the timestamps of all inputs match those that are expected by the vovserver.

To activate this functionality, use one of the following methods:

- Use the option `-n`
- Define the variable VOV_VW_NFS_PROTECTION

Examples:

```
% vw -n cp aa bb
% env VOV_VW_NFS_PROTECTION=1 vw cp aa bb
```

Here is a schema to try to explain the NFS cache problem better (and the need to wrap in vov):

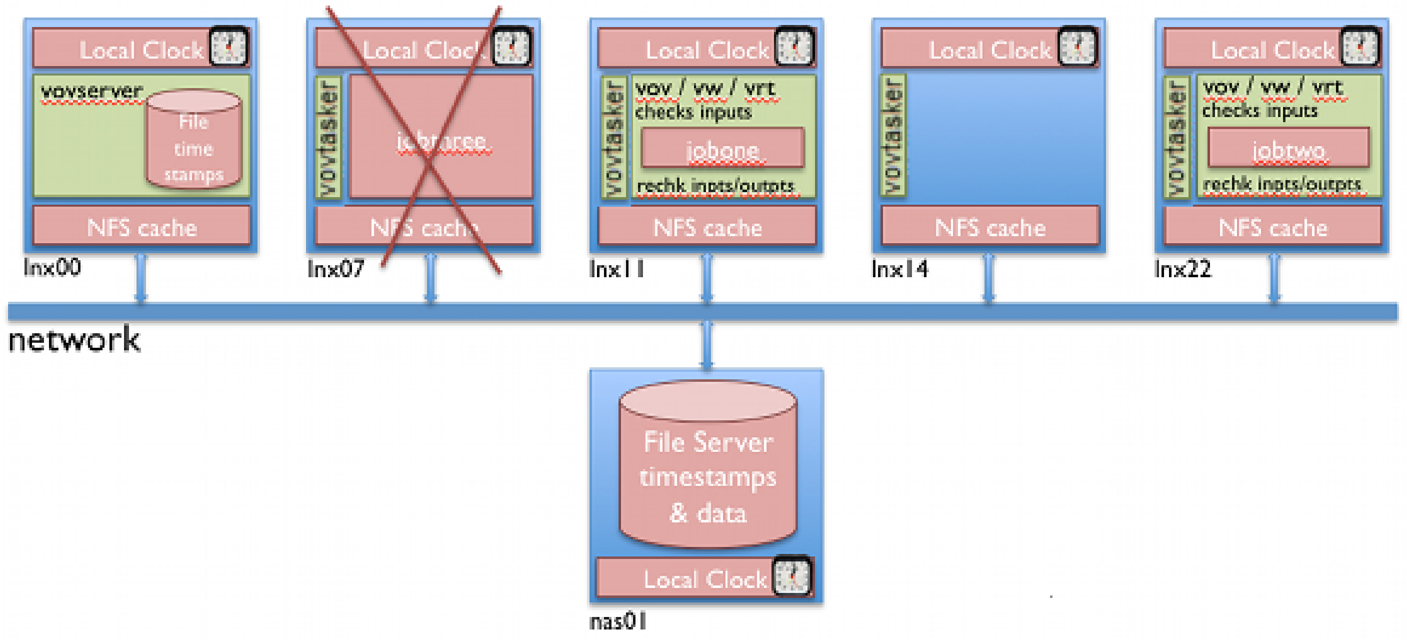


Figure 9:

Each UNIX host maintain its own local time, and an NFS cache which has the timestamps of the files. The cache stays valid for a few seconds before a file info is expired.

As long as we stay within the same machine, the file timestamps are correct to all the processes on that one machine. It is only when you have two processes back-to-back each running on a different machine that you may experience problems.

Let's first talk of a flow with two wrapped jobs: "jobone" and "jobtwo" which runs on lnx11 and lnx22. The vovtasker is just a remote execution process. It does not check any timestamps. It just cd to the directory, sets the environment, and starts the command line (a vov mytool).

When the vov wrapper wrapping jobone starts on lnx11:

1. It checks the inputs of jobone
2. It runs jobone
3. After the process jobone completes, it rechecks the inputs of jobone (they should not have changed) and then checks the outputs (they must all exist with timestamps greater than start of the job)

All this happens within one machine (lnx11) so we're safe.

Then the vov wrapper on lnx22 is starting jobtwo:

1. It checks the inputs of jobtwo
2. It runs jobtwo
3. After the process jobtwo completes, it rechecks the inputs of jobtwo (they should not have changed) and then checks the outputs (they must all exist with timestamps greater than start of the job)

When VOV_VW_NFS_PROTECTION is defined, the vov/vrt/vw wrappers wait 10 seconds and retry 6 times before finally exiting with an error if the problem is still present. That 60 seconds wait is sufficient for all the caches to synchronize over the network.

When the job is not wrapped (like jobthree on lnx07), the timestamps are checked by the vovserver itself. When the vovserver is on a different machine, it is exposed to NFS cache bugs. Always wrap your jobs with a vov or a vrt or vw if you want to avoid NFS cache problems. FlowTracer cannot wrap all jobs by default as it does not know which dependency method is required by the user, or the user may want to run extremely short jobs on the same machine with no overhead penalty.

Altair FlowTracer and ClearCase

Here is a list of suggestions to keep in mind when working in a ClearCase environment.

- The most important tool in ClearCase is called *cleartool*. Most users create an alias called "ct" or "cl". We only use the full name cleartool.
- Add the path to cleartool to the BASE environment. It often is in `/usr/atria/bin`.
- We support "single-view" projects. You have to start the server **AFTER** setting the view.

```
% cleartool setview  MYVIEWNAME
% vovproject start MYPROJECTNAME
```

- You need to know that to execute a command in a view you can use the following syntax:

  ```
  % cleartool setview -exec "COMMAND"  MYVIEWNAME
  ```

  This is what we do in vovtaskerstartup to start a vovtasker in the requested view.

Version Extended and View Extended Dependencies

For files in the VOBS, FlowTracer can track both.

- The version extended name (Example: /aa@@/main/3). These files will never change timestamp.
- The view extended name (Example: /view/MYVIEW/aa These files may change timestamp.

This capability is activated by setting the variable VOVCLEARCASE to the value ENABLED. We recommend you do this in the `setup.tcl` file.

```
# Add this line to the setup.tcl file.
setenv VOVCLEARCASE "ENABLED"
```

Regular Expressions in Selection Rules

The operators ~, !~, ~~ and !~~, which perform string matching, operate with regular expressions (Tcl 8.0). These expressions use the ed syntax. For a complete description of the syntax, invoke one of the following commands:

```
% man -s n regexp
% man ed
```

The following is a short summary of the syntax for regular expressions.

- The period "." stands for any character.
- The asterisk "*" stands for zero or more repetitions of the previous expression.
- The plus "+" stands for one or more repetitions of the previous expression.
- A backslash \ can be used to escape a literal period, asterisk or plus sign.

Examples

The regular expression ".*" matches every string (any character repeated any number of times).

If you want to make a set of all the files whose name contains .std., specify the regular expression "\.std\..*", in which "\." matches the period.

Time Specifications

Some VOV procedures and commands accept as input a time specification, which is a string that contains a mixture of digits and the letters **s m h d w**.

The letters are defined as follows:

Specification	Explanation
a	Seconds (default)
m	Minutes
h	Hours
d	Days (24 hours)
w	Weeks (168=7*24 hours)

The time specifications are case insensitive. The **d** and **w** specifications ignore that with daylight-saving some days may be 23 hours and other days may be 25 hours.

Examples of TimeSpecs

Specification	Explanation
60	60 seconds
2M	2 minutes, i.e. 120 seconds
3h30m	3 hours and 30 minutes, i.e. 12600 seconds

You can convert a time specification to seconds with the Tcl procedure `VovParseTimeSpec`. Conversely, you can convert an integer to a time specification, but with some loss of precision, with the procedure `vtk_time_pp`.

Some utilities (such as `ftlm_batch_report`) require a time interval specification. Time intervals can be expressed as follows:

- past hour
- today
- yesterday
- this week
- last week
- past week
- this month
- this month full
- last month
- past month
- past 30days
- this quarter
- last quarter
- this year
- last year
- YYYY, such as 2016, would be the entire year of 2016

- YYYYMM, such as 201016, which would be the month of December in 2016
- YYYYMMDD, such as 20100116, which would be Jan 15 2016
- YYYYwWW, such as 2017w4, which would be week 4 in year 2017
- Month YYYY. Example: Sep 2017
- start-finish, where on each side of the '-' is a timestamp specification that is parsed by VovScanClock. Example: 20090101-20090301

The conversion to a start-end pair is performed by the Tcl procedure `VovDate::computeSymbolicInterval`

Configuration .tcl Files

Configuration of FlowTracer on a per-project basis is handled by four .tcl files: policy.tcl, security.tcl, resources.tcl, and taskers.tcl.

policy.tcl

The `policy.tcl` file sets various server variables, such as the max number of Smart Sets, or max jobs in parallel. The syntax is as follows:

```
vtk_user_set  <user> -weight  <n> -defaultpriority <n> -maxpriority <n>
vtk_group_set <group> -weight <n> -users { <list_of_users> }
vtk_swd_set <symbolic> <path>
```

security.tcl

`security.tcl` defines who has access to the FlowTracer server. The main procedure used to define the "security rule" is:

```
vtk_security <username>|-group <vovusergroup> <securityLevel> <hostlist1> ...
```

- <username> is the login name of a user
- <vovusergroup> is the name of a VovUserGroups
- <securityLevel> is either USER, LEADER, or ADMIN, case insensitive
- <hostList> is a list of one or more of <hostName>, where <hostName> is either the name of a host, + meaning any host, or - meaning no host.

resources.tcl

`resources.tcl` defines both the rules to dispatch jobs based on priorities, as well as the number of licences available for tools used within your flow. The syntax is as follows:

```
vtk_tools_map <licence> <NUMBER> "rule"
vtk_resources_map <licence> <NUMBER> "rule"
vtk_resources_map <PRIORITY> <NUMBER>
vtk_toolmap_set <licence> <NUMBER> "rule"
vtk_resourcemap_set <User:> <NUMBER>
vtk_resourcemap_set <Group:> <NUMBER>
```

taskers.tcl

The `taskers.tcl` file defines the taskers for FlowTracer to use. The syntax is:

```
vtk_tasker_define hostname [options]
```

Legal Notices

Intellectual Property Rights Notice

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

Altair HyperWorks®, a Design & Simulation Platform

Altair® AcuSolve® ©1997-2025

Altair® Activate®©1989-2025

Altair® Automated Reporting Director™ ©2008-2022

Altair® Battery Damage Identifier™ ©2019-2025

Altair® CFD™ ©1990-2025

Altair Compose®©2007-2025

Altair® ConnectMe™ ©2014-2025

Altair® DesignAI™ ©2022-2025

Altair® DSim® ©2024-2025

Altair® DSim® Cloud ©2024-2025

Altair® DSim® Cloud CLI ©2024-2025

Altair® DSim® Studio ©2024-2025

Altair® EDEM™ ©2005-2025

Altair® EEvision™ ©2018-2025

Altair HPCWorks®, a HPC & Cloud Platform

Third Party Software Licenses

For a complete list of Altair Accelerator Third Party Software Licenses, please click here.

Technical Support

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

Altair One Customer Portal

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

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

Altair Training Classes

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

Index